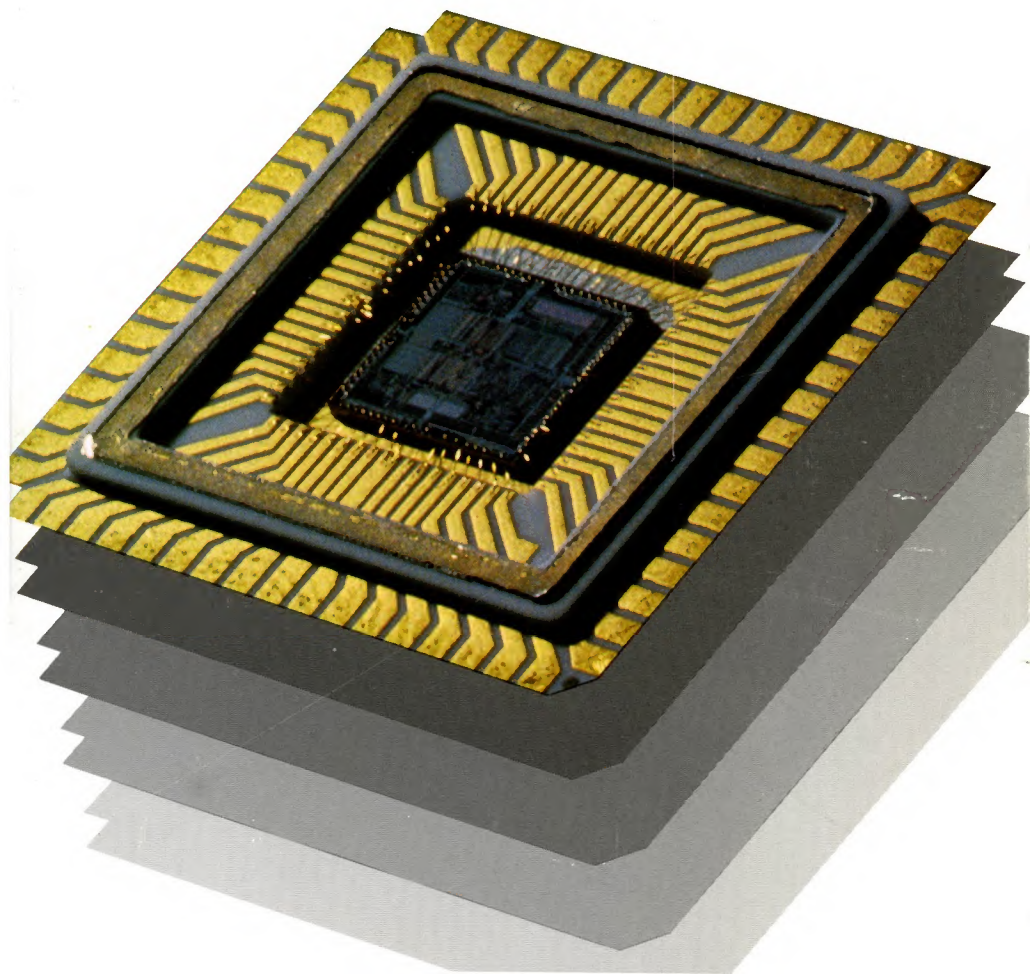




# NCR/32 GENERAL INFORMATION



## **PRELIMINARY**

This document contains the latest information available at the time of publication. However, NCR reserves the right to modify the contents of this material at any time. This data is supplied "As-Is"; NCR makes no warranty, express or implied, including but not limited to any implied warranty or merchantability or fitness for a particular purpose. Also, all features, functions and operations described herein may not be marketed in all parts of the world. Therefore, before using this document, consult your NCR Microelectronics Division Representative or NCR Microelectronics Division VLSI Product Marketing for the information that is applicable and current.

This is a revised edition of the General Information Manual. Changes made since the last edition are identified by bars in the applicable page margins.

## GENERAL INFORMATION MANUAL

## CONTENTS

<b>GENERAL INFORMATION.</b>	<b>1-1</b>
INTRODUCTION.	1-1
<b>SYSTEM ARCHITECTURE.</b>	<b>2-1</b>
Chip Capabilities.	2-1
NCR/32-000 Central Processor Chip (CPC).	2-1
NCR/32-010 Address Translation Chip (ATC).	2-4
NCR/32 System Structure.	2-4
Main Memory.	2-4
Scratch Pad.	2-5
ISU Memory.	2-5
NCR/32 System Addressing.	2-5
System Clock.	2-5
Processor Bus Structure.	2-7
ISU Bus.	2-7
PM Bus.	2-7
PMCHK Bus.	2-7
PM Bus Priority Control Logic.	2-7
Address Translation Function.	2-8
CPC Programming Model.	2-9
Microinstruction Set.	2-12
<b>PROCESSOR-MEMORY BUS.</b>	<b>3-1</b>
General.	3-2
Logic Conventions.	3-2
Common PM Bus Signals.	3-2
External Register Messages.	3-3
External Register Transfer Signal.	3-3
External Register Message Transfers.	3-4
Real Memory Messages.	3-4
Real Memory Transfer Signals.	3-4
Real Memory Message Transfers.	3-6
Virtual Memory and Memory Refresh Messages.	3-9
Processor PM Bus Transfers.	3-9
Processor Signals Related to PM Bus Transfers.	3-9
Processor PM Bus Access.	3-11
CPC Access to the PM Bus.	3-11
Special Request PM Bus Access.	3-11

# GENERAL INFORMATION MANUAL

## CONTENTS (CONTINUED)

Double Error Detection.....	3-12
ATC Intervention During Message Transfers.....	3-12
Virtual Memory Operations.....	3-12
Real Memory Operations.....	3-12
ATC Refresh Operations.....	3-14
PM Bus Timing.....	3-14
Virtual Message Transfer Timing.....	3-14
Virtual Memory Fetch Timing.....	3-15
Virtual Memory Full Store Timing.....	3-17
Virtual Memory Partial Store Timing.....	3-17
Real Memory Transfer Timing.....	3-20
CPC Real Memory Fetch Timing.....	3-20
CPC Real Memory Full Store Timing.....	3-22
CPC Real Memory Partial Store Timing.....	3-22
External Register (ERU) Timing.....	3-25
Processor Interrupt Message.....	3-26
Memory Interface.....	3-26
<b>CENTRAL PROCESSOR CHIP (CPC)</b> .....	<b>4-1</b>
CPC Overview .....	4-1
<b>SIGNAL DESCRIPTION</b> .....	<b>4-1</b>
<b>DATA ORGANIZATION IN MEMORY</b> .....	<b>4-7</b>
ISU Memory .....	4-7
Main Memory .....	4-8
<b>THREE-STAGE PIPELINE</b> .....	<b>4-9</b>
<b>ARITHMETIC LOGIC UNIT (ALU)</b> .....	<b>4-9</b>
Arithmetic Operations .....	4-10
Special Decimal ALU Logic.....	4-10
<b>REGISTER DESCRIPTION</b> .....	<b>4-11</b>
Register Storage Unit (RSU) .....	4-11
External Registers (ERUs) .....	4-13
Internal Register Unit (IRU) .....	4-14
Jump Registers (IRU0-IRU7) .....	4-14



## GENERAL INFORMATION MANUAL

## CONTENTS (CONTINUED)

Restore Field (IRU8) . . . . .	4-16
State Register (IRU9) . . . . .	4-17
Bits 1-5 — Field Array Bits . . . . .	4-18
Bits 7-10 — Byte Write Tag Bits . . . . .	4-18
Bits 11-14 — RSU Address Bits . . . . .	4-19
Bits 15-16 — Protection Check Code Bits . . . . .	4-19
Indicator Array (IRU16) . . . . .	4-20
Virtual Indicators (IRU17) . . . . .	4-22
Tally Register (IRU18) . . . . .	4-22
Operand Pointers (IRU24) . . . . .	4-23
Stack Pointer (IRU26) . . . . .	4-24
Setup Registers . . . . .	4-24
Setup Register #1 (IRU19) . . . . .	4-24
Setup Register #2 (IRU20) . . . . .	4-25
Setup Register #3 (IRU-10) . . . . .	4-25
Setup Register #4 (IRU11) . . . . .	4-25
Setup Register #5 (IRU18) . . . . .	4-25
Control Array #1 (IRU27) . . . . .	4-26
MARS6 Write Tags (IRU28) . . . . .	4-27
 WRITE TAG OPERATION . . . . .	 4-28
 SCRATCH PAD ACCESS . . . . .	 4-28
Access Via the Operand Pointers . . . . .	4-29
Access Via the Stack Pointer . . . . .	4-30
Scratch Pad Access Via ERU Registers . . . . .	4-31
Scratch Pad Access Via FL and SL . . . . .	4-31
 INTERRUPTS/TRAPS . . . . .	 4-33
Interrupt/Trap Recognition . . . . .	4-33
Interrupt/Trap Servicing . . . . .	4-33
Saving the Machine State . . . . .	4-34
Restoring From Interrupts/Traps . . . . .	4-34
 PROCESSOR STATE DURING RESET . . . . .	 4-35

## GENERAL INFORMATION MANUAL

### CONTENTS (CONTINUED)

PM BUS ACCESS .....	4-35
SETUP ASSIST .....	4-36
Setup Register Applications .....	4-36
Setup Register #1 Application .....	4-36
Setup Register #2 Application .....	4-37
Setup Register #3 Application .....	4-38
Setup Register #4 Application .....	4-38
Setup Register #5 Application .....	4-39
Scratch Pad Virtual Machine Operation .....	4-44
Operand Pointer #1 .....	4-44
Operand Pointer #2 .....	4-44
Stack Pointer .....	4-46
Map Indicator Logic .....	4-46
PROGRAMMING CONSIDERATIONS .....	4-46
Field Operands .....	4-46
Single Field Operand Instructions .....	4-47
Multiple Field Operand Instructions .....	4-48
Fetching From ISU .....	4-49
Delayed Jumps .....	4-50
Lock On Fetch .....	4-50
Store Operations .....	4-50
Fetch Operations .....	4-51
PROGRAMMING RESTRICTIONS .....	4-51
ADDRESS TRANSLATION CHIP (ATC) .....	5-1
ATC FUNCTIONAL DESCRIPTION .....	5-7
Memory Operations .....	5-7
Virtual Memory Operations (Address Translation) .....	5-8
Real Memory Operations .....	5-9
Memory Refresh Operation .....	5-9
Error Check/Correction and Syndrome Bit Generation (ECC) .....	5-10
ECC Generation During Memory Store .....	5-11

## GENERAL INFORMATION MANUAL

## CONTENTS (CONTINUED)

Error Check and Correction During Memory Fetch . . . . .	5-11
Multi-Work Fetch/Correction . . . . .	5-12
External Register Unit Operations . . . . .	5-12
Time Of Day/Interval Monitoring . . . . .	5-13
Virtual Address Monitoring . . . . .	5-13
Breakpoint (Fetch for Execute) Monitor — CA2 . . . . .	5-14
Address Monitor (Stores) — CA3 . . . . .	5-14
Fetch for Execute Monitor (Stores) — CA3,4 . . . . .	5-14
Trace (Store) Monitor — CA8 . . . . .	5-14
DYNAMIC ADDRESS TRANSLATION UNIT . . . . .	5-14
Overview . . . . .	5-18
Operation . . . . .	5-18
Translation . . . . .	5-18
Memory Protection . . . . .	5-19
Invalid Register (IR) . . . . .	5-20
Changed Page (CP) . . . . .	5-20
Register Referenced (RR) . . . . .	5-20
Protection . . . . .	5-21
Page Frame Number . . . . .	5-21
Virtual Page Number . . . . .	5-22
Interrupts . . . . .	5-22
EXTERNAL REGISTER DEFINITIONS . . . . .	5-22
Special Purpose ERUs . . . . .	5-23
Control Array #2 . . . . .	5-23
Control Array Definition . . . . .	5-24
Interrupt/Trap Array . . . . .	5-26
Trap and Interrupt Operation . . . . .	5-26
Trap and Interrupt Definition . . . . .	5-27
Interrupt Mask Register (IMR) . . . . .	5-30
Interval Timer/Monitor Register (ITMR) . . . . .	5-30
Time-Of-Day Register/Counter (TOD) . . . . .	5-30
Address Monitor Register (AMR) . . . . .	5-31
Bus Interrupt Register (BIN) . . . . .	5-31
Syndrome Register (SR) . . . . .	5-31

## GENERAL INFORMATION MANUAL

### CONTENTS (CONTINUED)

Memory Data/Processor Data Register (MD/PD) . . . . .	5-32
Virtual Operation ERUs . . . . .	5-32
Virtual Address Register (VAR) . . . . .	5-32
Real Address Register (RAR) . . . . .	5-33
Descriptor Data Register (DDR) . . . . .	5-34
Associative Memory and Page Descriptor Registers . . . . .	5-35
ASSOCIATIVE MEMORY COMMANDS . . . . .	5-35
Write Page Size (WPS) . . . . .	5-36
Read Page Size (RPS) . . . . .	5-36
Invalidate Associative Memory (IAM) . . . . .	5-36
Enable and Set Page Frame (ESPF) . . . . .	5-36
Restrictions . . . . .	5-37
Write Virtual Page (WVP) . . . . .	5-37
Read Page Frame (RPF) . . . . .	5-38
Write and Set Page Frame (WSPF) . . . . .	5-38
Restriction . . . . .	5-39
Clear Associative Memory (CAM) . . . . .	5-39
Purge Selective (PS) . . . . .	5-39
Restrictions . . . . .	5-40
Write Purge Mask (WPM) . . . . .	5-40
Read Purge Mask (RPM) . . . . .	5-41
Purge Selective with Mask (PSM) . . . . .	5-41
Restrictions . . . . .	5-42
Read Virtual Address (RVA) . . . . .	5-42
Read Real Address (RRA) . . . . .	5-42
Translate Virtual Address (TVA) . . . . .	5-44
Restrictions . . . . .	5-45
ATC STATE OPERATION . . . . .	5-45
State Flow . . . . .	5-47
SPECIAL ATC CONSIDERATIONS . . . . .	5-47
PM Bus Contention . . . . .	5-47
Refresh . . . . .	5-47
Time-Of-Day . . . . .	5-48

## GENERAL INFORMATION MANUAL

## CONTENTS (CONTINUED)

Bus Interrupt Register Interrupts. . . . .	5-48
Associative Memory Command Sequencing. . . . .	5-48
Monitor Operations. . . . .	5-48
ECC Disable. . . . .	5-48
ECC Generate/Syndrome Register. . . . .	5-49
Real Address Register Byte/Descriptor Data Register. . . . .	5-49
24/32 Bit Operations. . . . .	5-49
Associative Memory Results. . . . .	5-49
 TIMING CYCLE DESCRIPTIONS. . . . .	 5-49
Real Memory Operations. . . . .	5-49
Real Full Store. . . . .	5-50
Real Partial Store. . . . .	5-50
Real Fetch. . . . .	5-50
Virtual Memory Operations. . . . .	5-51
Virtual Full Store (CA9=0). . . . .	5-51
Virtual Partial Store (CA9=0). . . . .	5-52
Virtual Fetch (CA9=0). . . . .	5-52
Virtual Full Store (CA9=1). . . . .	5-53
Virtual Partial Store (CA9=1). . . . .	5-53
Virtual Fetch (CA9=1). . . . .	5-53
Refresh Operation. . . . .	5-54
 MICROINSTRUCTION SET. . . . .	 6-1
Microinstruction Set Format. . . . .	6-1
L Field Function. . . . .	6-1
K Field Function. . . . .	6-1
J Field Function. . . . .	6-2
I Field Function. . . . .	6-2
H Field Functions. . . . .	6-2
G Field Functions. . . . .	6-2
Instruction Nomenclature. . . . .	6-2

## GENERAL INFORMATION MANUAL

### CONTENTS (CONTINUED)

Instruction Operands .....	6-4
Full Word Operands .....	6-4
Half Word Operands .....	6-5
Byte Operands .....	6-5
Condition Selector .....	6-6
Field Operands .....	6-6
Single Field Operand Instructions .....	6-7
Multiple Field Operand Instructions .....	6-7
Literal Operands .....	6-8
Four Bit Literal .....	6-8
Eight Bit Literal .....	6-8
Sixteen Bit Literal .....	6-9
Digit Operands .....	6-9
Instruction Descriptions .....	6-9
Memory Instructions .....	6-9
Instruction Index by Function .....	6-9
Instruction Index by Op Code .....	6-15
Instruction Index by Mnemonic .....	6-20
<b>APPENDICES .....</b>	<b>A-1</b>
Appendix A, Glossary of Terms .....	A-1
Appendix B, Setup Flows .....	B-1
Appendix C, Breakpoint Operation .....	C-1
Appendix D, Memory Retries .....	D-1
Appendix E, Fetching from ISU .....	E-1
Appendix F, Non-Interruptible Instructions .....	F-1
Appendix G, Array Matrices .....	G-1
Appendix H, PBCD and UBCD Setting .....	H-1
Appendix I, Instruction Emulation Example .....	I-1

## CHAPTER I GENERAL INFORMATION

### CONTENTS

INTRODUCTION.....	1-1
MICROPROGRAMMING DESIGN.....	1-2
Microprogramming Definition.....	1-2
Microcontrol Section Organization.....	1-3
Architectural Features.....	1-4
CPC Chip Function.....	1-4
Instruction-set Partitioning.....	1-5
Performance Considerations.....	1-6
CPC Design.....	1-6
Other Chipset Features.....	1-6

## CHAPTER I

# GENERAL INFORMATION

The NCR/32 VLSI chipset introduces a new generation of programmable building blocks for the implementation of high-performance digital systems. This new generation combines the best features of cell library technology and microprocessor programmability with flexible, 32-bit power. For you, the system designer, external microprogrammability means that you can instruct the system both in what to do and how to do it. The NCR/32 family makes VLSI technology cost-effective in mainframe computer replacement for the first time.

Traditionally, the use of VLSI has been inversely proportional to the performance of a given computer system. The extremely high speeds, and the associated power dissipation, of common bipolar semiconductor technologies made VLSI devices of more than a few thousand gates impractical. In addition, packaging technologies had not been developed to provide sufficient interconnect capacity.

Several strategies are available today to increase the impact of VLSI in high-performance systems. NCR has adapted the design concepts learned from three product generations of large mainframe computers to the potentials offered by current VLSI technology. The NCR/32 semiconductor family provides new VLSI mechanisms for off-chip microcoding, instruction-set partitioning, and microcode primitives. These features offer a significant increase in performance, as well as substantially higher levels of integration.

The approach of maximizing synergism between NCR systems expertise and in-house microelectronics technology is the cornerstone of NCR's emergence as a leading merchant semiconductor vendor. We are now entering an expanding commercial market where NCR expects increased revenue and profit from our past R&D and capital equipment investments. This market also represents a fast-growing high-technology business that will keep us responsive to the demand for high quality and competitive prices.

All NCR Microelectronics products embody a three-pronged strategy for serving our customers. First, we are concentrating on MOS technology as the implementation vehicle for all products. Second, we specialize in quick-turn fabrication for semi-custom and custom logic products. And third, we emphasize custom solutions through microprogramming and cell library logic. NCR has selected



penetration of the microelectronics market as a fundamental strategy for continued corporate growth and investment in the future.

### MICROPROCESSOR DESIGN

Figure 1-1 shows the main functions of a computer and its critical information flows. The control (CTL) section is responsible for determining the order, timing, and direction of information flow between functional blocks. When executing a stored program, the CTL section puts in sequence the operations of FETCH, DECODE, change controls, and change state.

In first generation microprocessor designs, the control section was implemented in the Microprocessor Unit (MPU) as random logic, with flip-flops and timers sequencing the execution of MPU instructions. Late in the second generation, some MPU designs began to apply microprogramming techniques internally; and most third generation 16/32 bit MPU products incorporate some form of internal microprogramming (to conserve space and increase the number of features). The NCR/32 family is the first commercially available 32-bit MPU to offer external microprogram capability to the user.

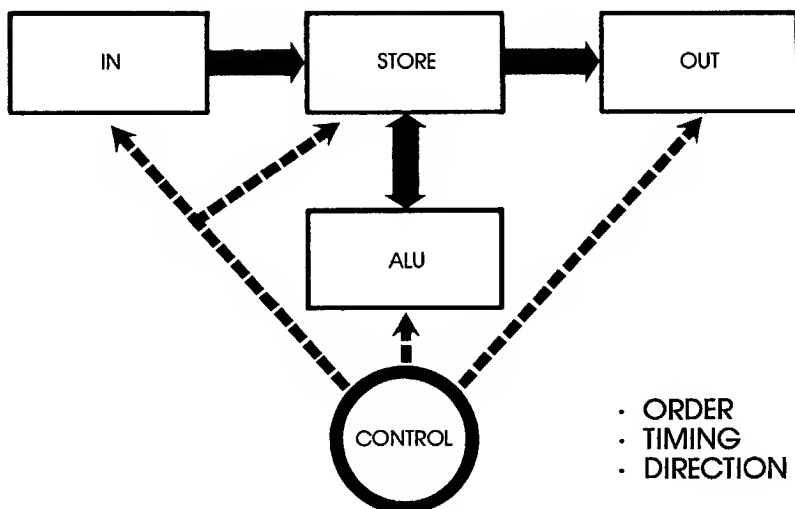


Figure 1-1 Computer Functions

### MICROPROGRAMMING DEFINITION

To understand how microprogramming differs from programming an MPU, imagine that a program for an MPU directs the computer system in what to do while a microprogram tells the MPU how to do it. This capability, not traditionally available to the programmer,

is defined by Samir S. Husson in Microprogramming: Principles and Practices:

“Microprogramming is a technique for designing and implementing the control function of a system as a sequence of control signals, to interpret fixed or dynamically changing data processing functions. These control signals, organized on a word basis and stored in a . . . control memory, represent the states of the signals which control the flow of information between the executing functions and the orderly transition between these signal states.”

This means that any machine instruction of an MPU is essentially a “closed” subroutine executed by microinstructions fetched from the closed store.

### Microcontrol Section Organization

Several design approaches are possible for implementing a microcontrol subsystem, and two types of organization have been developed: horizontal and vertical.

In order to achieve maximum performance with a given data path organization, the microcontrol section must provide a high degree of parallelism. Since many control signals are not mutually exclusive, a very wide microinstruction word is required to specify the state of all control signals during a given microcycle. This wide microinstruction is called horizontal microcode (Figure 1-2). When it is feasible to encode the control signals less densely, the microword organization can be very narrow; this is called vertical microcode (also Figure 1-2). Because of the encoding inherent in the vertical microcoding, more words are required to execute the same function, and vertical control stores require larger address spaces. However, this allows reduced pin count and ease in debugging.

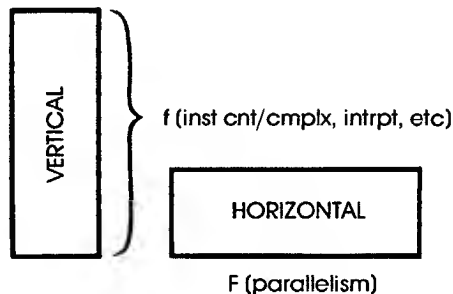


Figure 1-2

Microstore Organizations

## ARCHITECTURAL FEATURES

As shown in Figure 1-3 the NCR/32 chipset consists of functional building blocks which address the design problems of computer systems. Consisting of a microprogrammed processor (the CPC), a memory manager (the ATC), a series of performance booster circuits (such as the EAC), and an I/O controller (the SIC/SIT/SIR), the NCR/32 family can be used in a variety of applications. These range from mainframe and super-minicomputer emulation to dedicated control functions. Each building block can be used separately or in conjunction with other family members to provide a rich library of capabilities.

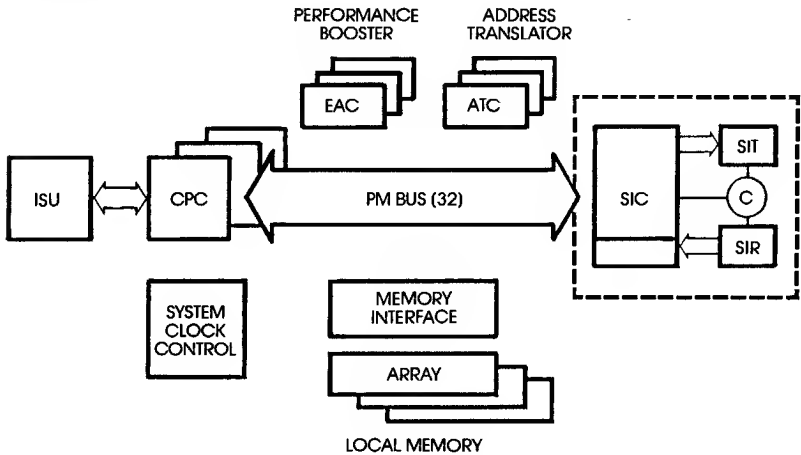


Figure 1-3 NCR/32 Family

### CPC Chip Function

Figure 1-4 illustrates the data path organization of the CPC and the pipeline implementation. The CPC uses two levels of microcoding. An off-chip vertical microinstruction is fetched from the Instruction Storage Unit (ISU) to begin execution of each microcycle. On-chip, the CPC uses a small horizontal control memory to provide the control signals necessary for data flow and execution. By using the two levels of microcoding and a three-stage pipeline, the CPC completes one external microinstruction approximately every 150 nanoseconds.

A 16-bit multiplexed data/address bus (ISUBUS) provides the communications path for vertical microinstructions into the CPC. Each ISU word is divided into fields (Figure 1-5) for easy decoding. The G field is used as the control store address for the on-chip horizontal microinstruction memory, selecting the control and information flow during the execute state of the pipeline. The H and I fields provide operand selection specification during certain micro-

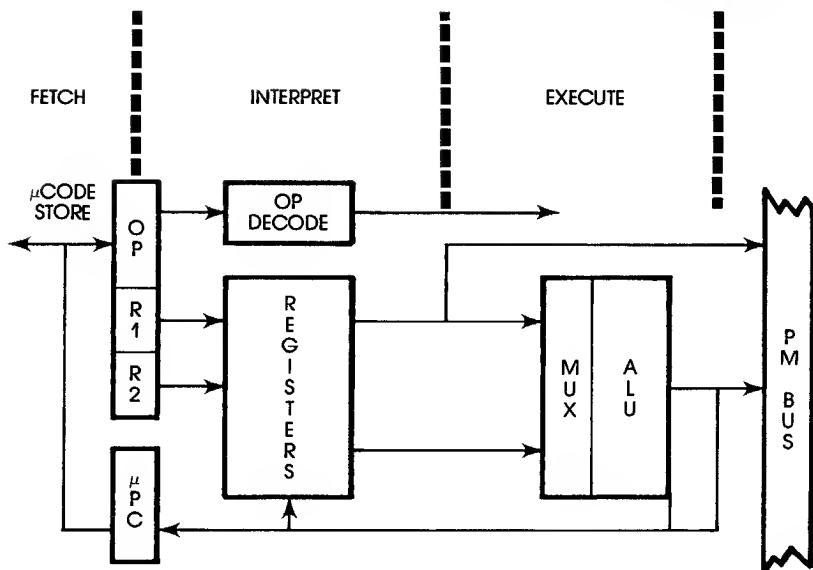


Figure 1-4 CPC Pipeline

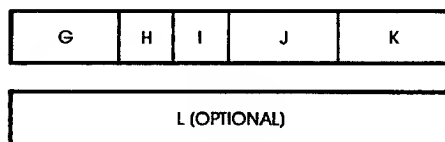


Figure 1-5 ISU Fields

instructions, when an extended op-code is needed or a non-RSU operation will be executed. The J and K fields are register specifiers that determine which operands are used for execution. The L field provides a 16-bit literal operand during some microinstructions.

### Instruction-set Partitioning

Microcode primitives are provided by special hardwired logic and internal registers that, in conjunction with the external scratch-pad memory, define the virtual machine architecture and decode virtual instructions. These primitives, called macroinstruction set-up commands, speed execution by replacing multiple microinstructions required to load registers with the contents of virtual instruction fields. Modifications can be made to the CPC on-chip to optimize this special logic for particular virtual machine emulation. Current implementations support IBM 370 and NCR mainframe instruction sets.

Instruction-set partitioning is implemented by the addition of special performance booster chips, such as the EAC. Depending on the performance boost required, these chips may monitor all in-

struction fetches from memory and control the CPC, or act as “slaves” to the CPC, receiving all operands and commands under CPC control. Several different combinations have been used to excellent advantages in equipment designs.

## PERFORMANCE CONSIDERATIONS

System performance can vary widely based on your particular implementation strategy. Each NCR/32 building block offers several choices you should evaluate while considering the best hardware/firmware trade-offs for your application. In designing your system, make sure you consider functional partitioning, memory utilization, and I/O interfacing. (These areas equate roughly to the degree of parallelism and amount of bandwidth inherent in your system design.)

### CPC Design

The CPC design is crucial to ISU organization and sequencing for your system. The 16-bit microinstruction word length and the internal control register define the minimum implementation required for an NCR/32 system. Beyond that, you have considerable flexibility in choosing the optimum scheme for your performance needs. Expanding the word length can provide additional control bits to assist in parallel execution of mutually exclusive functions. For emulation, performance can be improved dramatically by overlapping instruction fetch and decode, operand set-up, and execution. Similarly, supplementing the CPC control register and jump logic with external circuitry can increase the range of control decision and expand the total ISU addressing range.

### Other Chipset Features

When your system performance dictates specialized hardware to improve speed, the NCR/32 chipset offers building blocks that support functional partitioning of the execution task. The EAC provides specialized hardware for floating point arithmetic and can improve speeds by an order of magnitude compared to in-line firmware implementations. Careful instrumentation of dynamic system performance can identify the critical bottlenecks that specialized hardware can overcome to meet your system goals.

Your memory organization and performance can also vary widely, depending on the features that you use. Both ECC logic and address translation logic can be disabled to allow you to choose mechanisms for implementing these functions. Transfers of data over the PM bus can take one or more system cycles to complete, allowing you to control both the cost and the performance of your memory subsystem. Fast caching techniques can further improve

performance for specific applications.

Finally, I/O interfacing techniques can be optimized to suit your cost and performance goals. Using DMA block transfers will minimize CPC overhead, but will cost more for intelligent I/O control logic. Using the SIC/SIT/SIR subsystem will provide a high-bandwidth serial data link for low-cost peripheral interfaces. Other interfaces to popular I/O environments such as Multibus can be easily accommodated by NCR/32 family building blocks.

All of these choices provide greater flexibility in meeting the particular application requirements of your system design. In contrast to other microprocessor families, the NCR/32 family provides uncommitted system building blocks which allow customization, when necessary, to suit your needs. NCR is also committed to continued refinement and expansion of the entire NCR/32 family.

This General Information Manual provides the technical information you will need to help you determine if the NCR/32 family can fulfill your application needs. Additional technical details on electrical and mechanical features of the individual chips are provided in data sheets available from your local NCR Microelectronics sales representatives. Questions about particular application concerns are fully supported by our applications engineers.

## CHAPTER II

## SYSTEM ARCHITECTURE

### CONTENTS

Chip Capabilities. . . . .	2-1
NCR/32-000 Central Processor Chip (CPC). . . . .	2-1
NCR/32-010 Address Translation Chip (ATC). . . . .	2-4
NCR/32 System Structure. . . . .	2-4
Main Memory. . . . .	2-4
Scratch Pad. . . . .	2-5
ISU Memory. . . . .	2-5
NCR/32 System Addressing. . . . .	2-5
System Clock. . . . .	2-5
Processor Bus Structure. . . . .	2-7
ISU Bus. . . . .	2-7
PM Bus. . . . .	2-7
PMCHK Bus. . . . .	2-7
PM Bus Priority Control Logic. . . . .	2-7
Address Translation Function. . . . .	2-8
CPC Programming Model. . . . .	2-9
Microinstruction Set. . . . .	2-12

## CHAPTER II

# SYSTEM ARCHITECTURE

This chapter provides a general description of the NCR/32 Processor system. The features of the Central Processor Chip (CPC) and Address Translation Chip (ATC) are described, as well as how these two devices interact in a system. The CPC, ATC, and Main Memory are interfaced via the Processor-Memory Bus (PM Bus). The PM Bus is described in detail in Chapter III. Detailed descriptions of the CPC and ATC are provided in Chapters IV, V, and VI. Figure 2-1 shows a block diagram of the NCR/32 system.

### CHIP CAPABILITIES

The following paragraphs provide a general description of the CPC and ATC and are followed by discussion of the NCR/32 system.

#### NCR/32-000 Central Processor Chip (CPC)

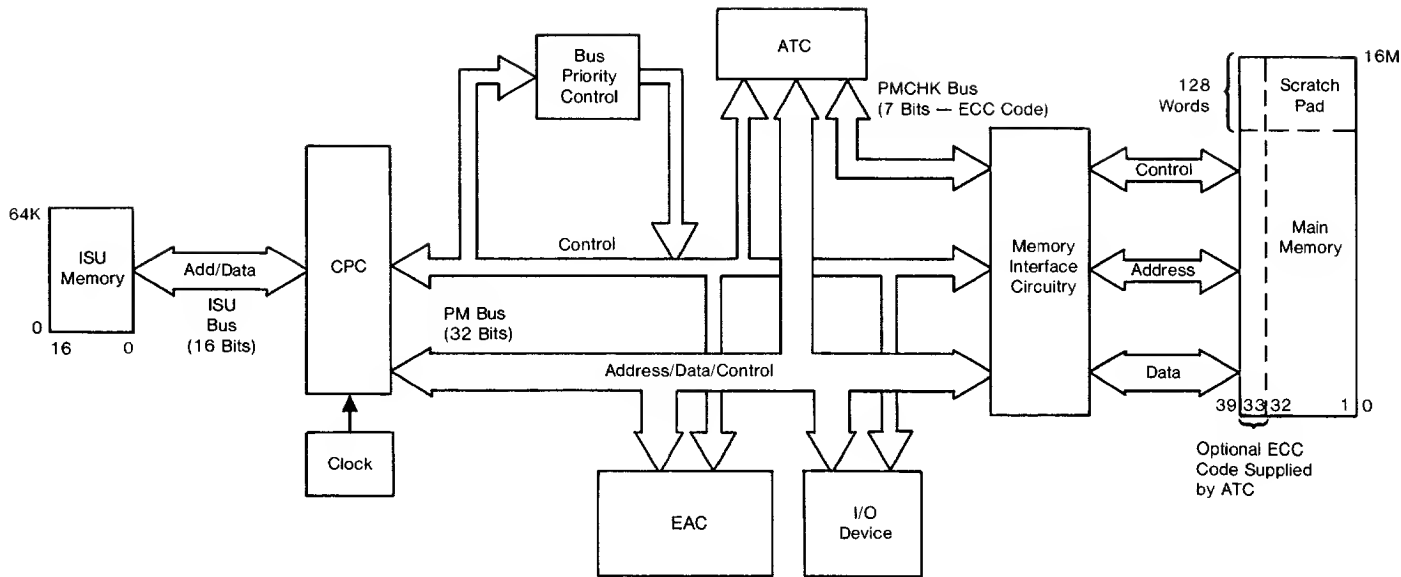
The Central Processor Chip (CPC) is a self-contained, 32-bit architecture, microprocessor element that provides the logic to execute an NCR/32 Processor user microinstruction program stored in the Instruction Storage Unit (ISU) memory. The CPC can operate at three levels of programming. In level one, the microcode instructions are fetched from ISU and executed directly to perform program functions (e.g., executing a high level language directly from microcode or performing a controller function like a graphics controller). In level two, groups of instructions in ISU are fetched to execute software instructions stored in main memory. This level is used when the CPC is emulating a virtual machine. The virtual machine instructions are located in main memory while the CPC microinstruction routines to emulate each virtual machine instruction are located in ISU. A third level of programming is available in which the software fetched by ISU microinstructions access microcode subroutines and look-up tables stored in the ISU.

A breakdown of a 32-bit word is shown in Figure 2-2.

The major features of the CPC include:

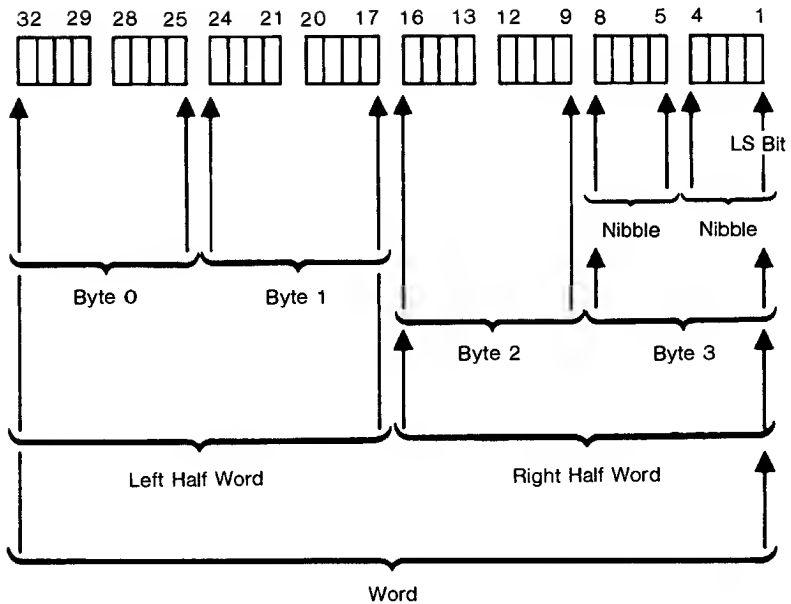
- True 32-bit internal/external architecture
- External microprogrammability





GIM2318A

Figure 2-1 NCR/32 System Configuration



GIM2103AAA

Figure 2-2 32-Bit Word Breakdown

- Two independent external data paths
  - 32-bit Processor-Memory Bus (PM Bus)
  - 16-bit Microinstruction Storage Bus (ISU Bus)
- Sixteen 32-bit true general purpose registers (RSU)
- 32-bit ALU
  - Digit (nibble), byte, halfword, word, and field (string) data types
  - Decimal, binary, and boolean operations
- Addressing Range
  - 4 Gigabytes of direct virtual memory
  - 16 Megabytes of direct real memory
  - 128 Kilobytes of direct microinstruction memory
- 179 microinstructions and variants with register to register format
- 95% of instructions execute in one clock cycle
- 3-stage microinstruction pipeline
- 8 addressable 16-bit microinstruction jump registers
- 3 main memory scratch pad pointers
- Special hardware for opcode cracking when emulating virtual machines
- NMOS silicon gate technology

## **NCR/32-010 Address Translation Chip (ATC)**

The Address Translation Chip (ATC) is an optional device which provides memory management assistance to the CPC. It contains an address translation unit, a memory data syndrome bit generator, syndrome bit checker and data correction logic, a time-of-day counter/register, memory refresh circuitry, and special registers available to the CPC.

The major features of the ATC include:

- Full support of virtual to real addressing including full or partial word stores
- Virtual address monitoring
- Sixteen address translation registers
- Supervisor/user modes with four levels of protection in each
- Memory refresh/scrubbing functions
- Syndrome (check) bit generation (ECC) for all memory stores
- Error check and correction for memory fetches
- Time-of-day and time interval monitoring
- Variable byte page sizes (1k, 2k, and 4k)

The ATC supports three types of memory operations:

1. Real memory operations generated external to the ATC.
2. Virtual memory operations (address translation plus resulting real memory operation).
3. Virtual equals real memory operations (real memory operation generated from untranslated virtual address).

## **NCR/32 SYSTEM STRUCTURE**

The NCR/32 Processor Family, when interfaced as a system, forms a general-purpose microprocessor system and virtual machine emulator. The CPC, through its vertical microinstruction set, supports virtually all mainframe opcode functions and data handling capabilities. This section provides a general functional description of how the NCR/32 Processor Family of chips may be assembled to function in a system.

### **Main Memory**

The Main Memory serves as a data and software storage area. It consists of RAM (typically Dynamic) chips connected to the CPC via the memory interface and PM Bus. The main memory is organized into 32-bit wide words (39 if ECC is used) and can be expanded to 16M bytes. The CPC scratch pad is located at the top of main memory.

**Scratch Pad**—The NCR/32 system architecture was heavily influenced by advances made in state-of-the-art software technology. High level languages as well as code produced from high level languages must run efficiently on new generation processor systems. Efficient high level language generation or emulation of high level language machines requires that parameters be easily transferred between and within software modules. The NCR/32 processor system allows this with a powerful yet easy to use scratch pad function.

A 128-word by 32-bit scratch pad is located at the top of main memory. Special addressing registers inside the CPC simplify the emulation of register and stack oriented machines. The scratch pad is intended to be used as operand stack and virtual registers for data manipulation and parameter passing. The first 64 words of scratch pad can be explicitly addressed with special memory (literal) instructions. They can also be addressed indirectly through operand pointer registers, as can all of the 128 words. These pointer registers are located in the CPC. See Chapter IV for a more complete description of the scratch pad.

### ISU Memory

The Instruction Storage Unit (ISU) contains the CPC user microcode programs and subroutines using the CPC 16-bit microcode instruction set. The ISU memory is organized as a 64k x 16 memory space and is addressed by the CPC via the ISU Bus.

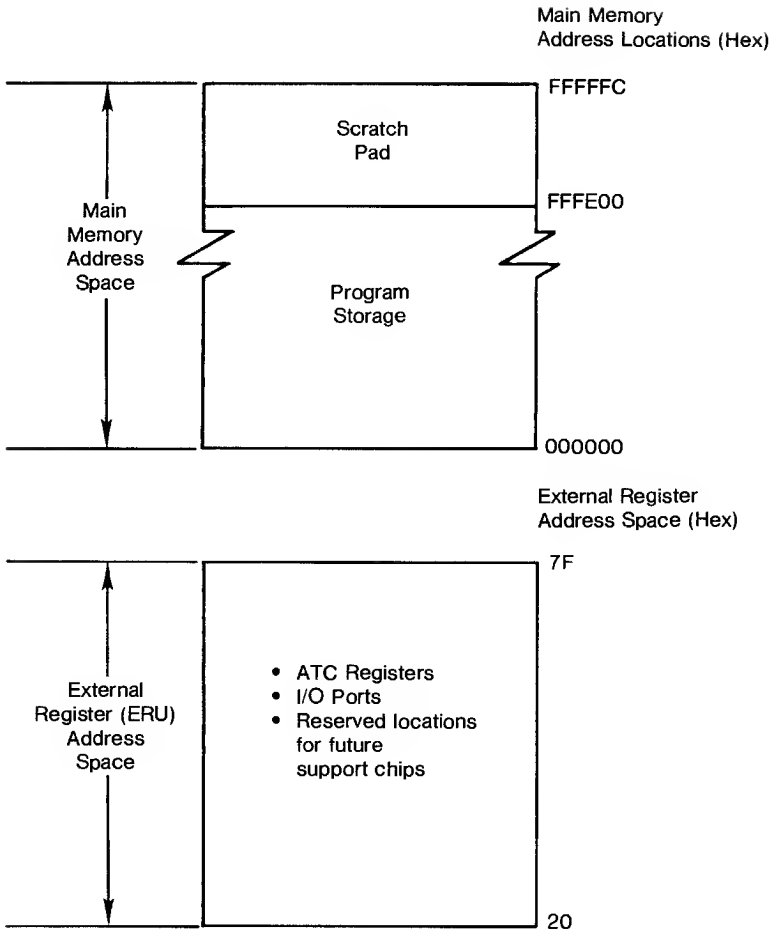
### NCR/32 System Addressing

The NCR/32 Family of chips, including the CPC and ATC, are part of a system that is interfaced via the PM Bus. CPC support chips are not located in the memory address space of the CPC (see Figure 2-3). Registers inside the ATC and other support chips in the system are referred to as External Registers since they are external to the CPC, and are accessed by a special set of single cycle external register transfer instructions. Details on the external register transfers are provided in Chapter III.

### System Clock

The system clock is a two-phase non-overlapping clock. The first half of each cycle is called phase 0, and the second half of each cycle is called phase 1. In text "X0" will be used to indicate phase 0, and "X1" to indicate phase 1.

When data transfer operations are initiated, the ISU Bus and/or the PM Bus transfer an address during X0, and the system control lines are set to address-related states. During X1 the data to be transferred is asserted on the bus, and the system control lines are set to data/read/write-related states.



## NOTES:

1. Accesses to/from the main memory address space use real and virtual memory fetch and Store instructions. See chapter VI—"Instruction Set" for details. The only exceptions are ERU transfers to scratch pad (through ERU locations 20-26 Hex). See chapter IV—"CPC" for details.
2. Accesses to/from the External Register Address Space use Transfer in External (TIE) and Transfer Out External (TOE) instructions. These are single cycle operations which use a special PM Bus control signal (EREP—External Register Enable/Permit) to distinguish them from transfers to/from main memory. The only exceptions are ERU transfers to Scratch Pad (through ERU locations 20-26 Hex) which assert real memory operations on the PM Bus.

GIM2321

Figure 2-3 System Address Partitioning on the PM Bus

When the Address Translation Chip (ATC) is performing a memory store operation, the ATC holds the data on the PM Bus for several system clock cycles until the destination device (memory) has received the data.

## Processor Bus Structure

The Processor Bus Structure connects the various devices in the NCR/32 Family as shown in Figure 2-1. The major busses, the ISU Bus and the PM Bus, each transfer address and data on the same lines. The PMCHK Bus transfers only the syndrome (check) bits.

**ISU Bus**—The 16-bit bi-directional ISU Bus connects the CPC to the Instruction Storage Unit. During X0 of the system clock the CPC asserts the address of the next microinstruction (covered in detail in Chapter IV) on the bus. During the following X1 the ISU asserts the addressed microinstruction on the ISU Bus, and the CPC loads it into an internal Instruction Register (IR).

**PM Bus**—The 32-bit bi-directional PM Bus interfaces the CPC to all devices in the system. Memory fetch and store operations occur in two basic steps. During X0 the address is asserted on the bus, then the following X1 the CPC reads the bus or asserts data on it. Fetch/store control signals establish which is to occur and latch the data into the appropriate register or memory location. The ATC controls the PM Bus in the same fashion as the CPC when the ATC is directed to transfer data. A complete description of the PM Bus is provided in Chapter III.

**PMCHK Bus**—The 7-bit bi-directional PMCHK Bus interconnects the ATC and main memory. It transfers the syndrome (check) bits that are used by the ECC logic in the ATC to ensure validity of the data transferred on the PM Bus.

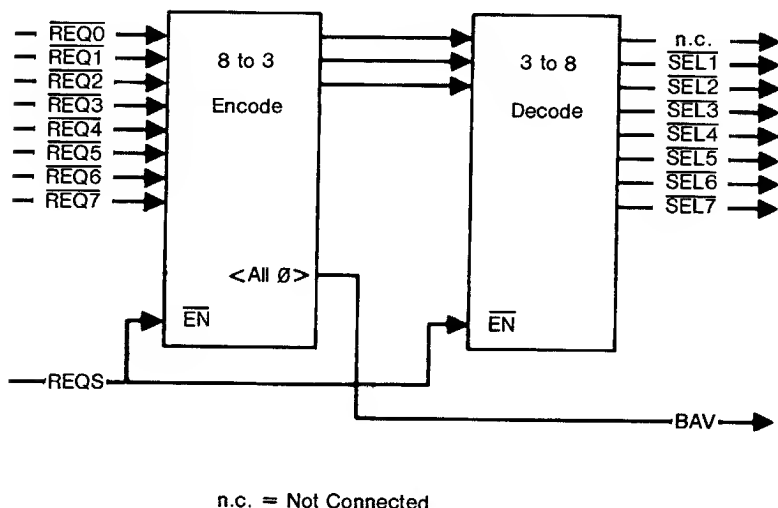
## PM Bus Priority Control Logic

The Bus priority logic is user-designed hardware that controls the time sharing (arbitration) of the PM Bus. It consists of circuitry that encodes during X0 the Request lines ( $\overline{REQ0}$  to  $\overline{REQn}$ ) from individual system chips requiring access to the PM Bus. During X1 the Bus Priority logic asserts the appropriate Select line ( $\overline{SEL1}$  to  $\overline{SELn}$ ) to enable the selected chip during the next clock cycle.  $\overline{SEL0}$ , reserved for the ATC, is not used because the ATC has highest priority and should be guaranteed the bus on the next cycle after asserting  $\overline{REQ0}$ . An example of a simplified eight-input version of the circuitry is shown in Figure 2-4.

The ATC typically is assigned the highest priority ( $\overline{\text{REQ0}}$ ) for PM Bus access. The lowest priority is normally assigned to the CPC.

The REQS line from the ATC must override all  $\overline{\text{REQn}}$  signals to give the PM Bus to the CPC on a cycle steal basis when required. In the Bus Priority logic an active REQS must force BAV active (high) and disable all  $\overline{\text{SELN}}$  signals. Otherwise, in the absence of a Bus request, the BAV signal should default to an active (high) state, allowing the CPC to have access to the bus.

A block diagram of a Bus priority circuit is shown in Figure 2-4.



Note: The circuitry in this figure provides for up to 8 devices to be connected to the PM Bus. If the CPC and ATC are the only devices then  $\overline{\text{REQ1}}$ - $\overline{\text{REQ7}}$  and  $\overline{\text{SEL1}}$ - $\overline{\text{SEL7}}$  are not used.

GIM2104AA

Figure 2-4. Bus Priority Logic (example circuit)

## Address Translation Function

The ATC adds memory access partitioning and virtual address translation capabilities to the CPC system. The ATC is completely programmable from the PM Bus, and the CPC and other active devices can direct the ATC to perform data transfers with error checks, perform just the error checks, or remain inactive.

The ATC performs two types of fetch/store message operations: Real Memory operations and Virtual Memory operations. When a

real memory fetch or store operation is initiated, the ATC performs no operation on the address, and the address is applied directly to memory followed by the data transfer.

When the ATC is active, memory store operations will begin during X1 and continue through as many cycles as required to meet the access time of the main memory RAMs. The ATC holds the data on the PM Bus until the memory interface asserts the DIE signal. The ATC also sends Byte Write Enables to gate the data into the addressed memory during store operations.

When a virtual memory fetch or store operation is initiated, the virtual address is latched into the ATC and translated into a real address during X0. The virtual to real address translation, in brief terms, consists of concatenating the virtual (relative) address to a reference point (page start) address previously loaded into the Page Frame Number (PFN) register and addressed by one of the Virtual Page Number (VPN) registers. If it is to be a store operation, while the translation is being accomplished the data to be transferred is clocked into the ATC during X1. During the next X0 the resulting real address is asserted on the PM Bus by the ATC and the data transfer is initiated the following X1.

During real and virtual memory fetch operations, the ATC performs automatic error checks (ECC) and corrects erroneous data if possible. When an error is uncorrectable the data is passed with an asserted Memory Data Enable/Error (MDEE) signal indicating that the data on the PM Bus is invalid. NOTE: The MDEE signal from the ATC is reserved for use by I/O devices on the PM Bus. The ATC informs the CPC of uncorrectable errors via a trap (TRAP line).

The ATC provides the necessary timing, signals, and data transfers for memory refresh operations to the memory interface. In addition, the Time-Of-Day Register/Counter (TOD) in the ATC may be read via the PM Bus or used to generate timed interval interrupts. When memory refresh operations are required, the TOD is used to determine when they are to occur.

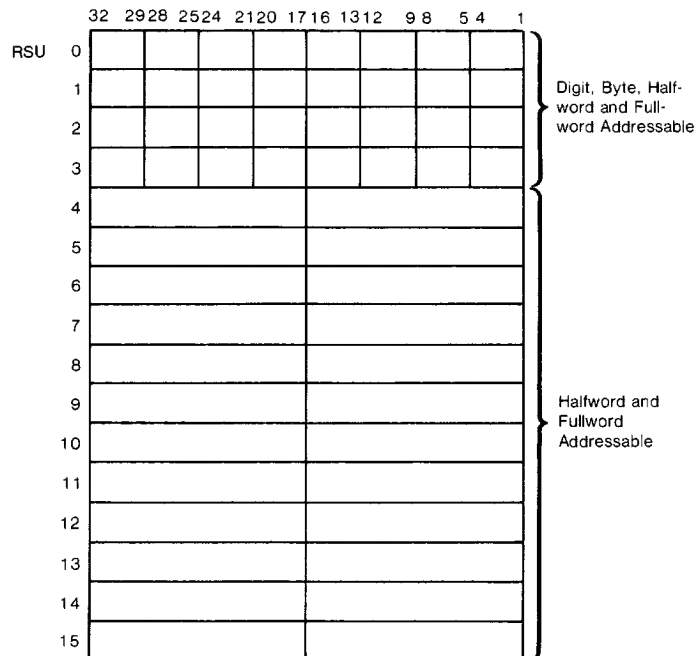
## CPC Programming Model

A very powerful set of data and address registers inside the CPC makes programming easier, faster, and more reliable. The Register Storage Unit (RSU) consists of sixteen 32-bit general purpose registers. All sixteen registers are word and halfword addressable while the first four are also byte addressable, as shown in Figure 2-5(A). In the general sense the RSU contains operands for manipulation (digit, byte, halfword, and word) by the CPC ALU and is a storage unit for data transfers to/from main memory.



2-10

(a) Register Storage Unit (RSU) as a general purpose register set



(b) Register Storage Unit (RSU) as Memory Assist Register Set (MARS) for memory or field instructions.

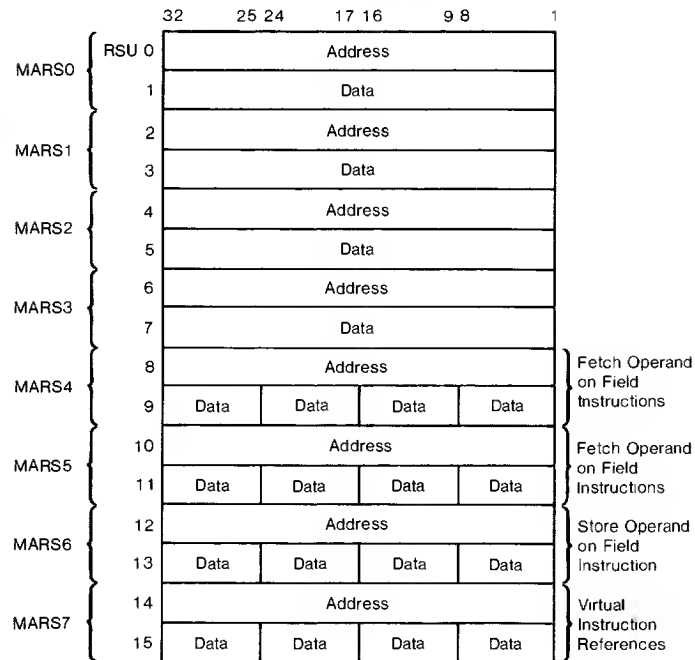


Figure 2-5 Programming Model

During field (string) operations the RSU locations 8-15 are organized into even-odd register pairs. When operating on field data the RSU is referred to as the MARS (Memory Assist Register Set) registers. The even register contains the field data address, while the associated odd register contains the field data. Special pointers, MARS byte pointers, automatically keep track of word boundaries and inform the CPC when the next field data word should be fetched from memory.

During virtual machine emulation two of the RSU registers function as Virtual Control Registers. RSU14 is the virtual machine program counter, and RSU15 is the virtual machine instruction register. A more complete description of the RSU is provided in Chapter IV. A model of the MARS registers is presented in Figure 2-5(B), and RSU characteristics are presented in Table 2-1.

RSU #	Word	Transfers Halfword	Byte *	Digit	MARS Function	Field Usage
0 1	X X	X X	X X	X X	MARS0 Addr MARS0 Data	
2 3	X X	X X	X X	X X	MARS1 Addr MARS1 Data	
4 5	X X	X X			MARS2 Addr MARS2 Data	
6 7	X X	X X			MARS3 Addr MARS3 Data	
8 9	X X	X X	X		MARS4 Addr MARS4 Data	Operand Fetch
10 11	X X	X X	X		MARS5 Addr MARS5 Data	Operand Fetch
12 13	X X	X X	X		MARS6 Addr MARS6 Data	Operand Store
14 15	X X	X X	X		MARS7 Addr MARS7 Data	Virtual Control Registers

\*Byte transfers on RSU 9, 11, 13, and 15 are during field operations only

GIMTE2320

Table 2-1 RSU Characteristics

## Microinstruction Set

In general, the 16-bit CPC microinstructions store operands in the RSU, perform operations on the operands, and place the result back into the RSU. Each microinstruction consists of an 8-bit opcode and either two 4-bit RSU register pointers, an 8-bit control code, or an 8-bit literal. Some microinstructions require an additional 16-bit literal.

The simplicity of the CPC lets most instruction execution follow the same pattern: (1) read one or two registers, (2) perform an operation with the contents, and (3) store the result in a register. This results in a significant reduction in decision making, and allows faster program execution.

The general categories of microinstructions in the CPC instruction set are: Memory Transfer, Logical, Arithmetic, Jump, and Special. The microinstructions, depending on their type, operate with 4-bit nibbles (digits), 8-bit bytes, 16-bit halfwords, 32-bit words, and fields of bytes, where a field consists of one to  $(64K-1)$  bytes.

The memory instructions include various combinations of real and virtual memory fetch and memory store operations with literal or register addresses. The instructions allow memory store operations of words, halfwords, or individual bytes by using the Byte Write Enable for each of the four bytes in a word.

The transfer instructions include byte, half word, word, and field (string) operations. A group of logical and arithmetic instructions are also included. During field operations the Tally Register inside the CPC keeps track of the remaining bytes in the field. The Tally Register Decrements to zero when all of the field bytes have been transferred.

The logical instructions perform the Boolean operations: OR, AND, and EXCLUSIVE OR on bytes, halfwords, words, and fields.

The arithmetic instructions include add, subtract, and shift, for various data types. They can operate with packed and unpacked decimal on bytes and fields, and in binary on bytes, halfwords, words, and fields.

The special instructions include setup commands, special load commands, and instructions for setting, resetting, and restoring flag bits.

The jump instructions can be either delayed, skip, or immediate operations. The delayed jumps allow the next two instructions to be executed before the jump. The skips jump the next two instructions if the conditions are met. The delayed and immediate jumps can be conditional or unconditional to direct, register, or relative addresses. There are also conditional returns to add program flexibility.

An immediate program branch (e.g., immediate jump) voids the

CPC 3-stage pipeline. The delayed jump increases efficiency by allowing the next two instructions already loaded in the pipeline to execute. The efficient programmer is able to maximize performance by using delayed jump and return instructions.

The memory fetch sequence is supported by an implementation which allows either performance optimization or coding simplification. A fetch microinstruction is required to initiate the memory fetch operation, and a RCV instruction to load the fetched data into a destination RSU. The efficient programmer is able to analyze each fetch sequence which requires more than two processor cycles to complete and code it uniquely (e.g., F, TW, RCV). The instructions between the FETCH and RCV must not reference the PM Bus or, of course, the data being fetched. The code conscious programmer merely develops a standard macro (FETCH-RCV) and lets the CPC automatically compensate for the actual number of cycles required, determined by the assertion of DIE.

## CHAPTER III PROCESSOR-MEMORY BUS

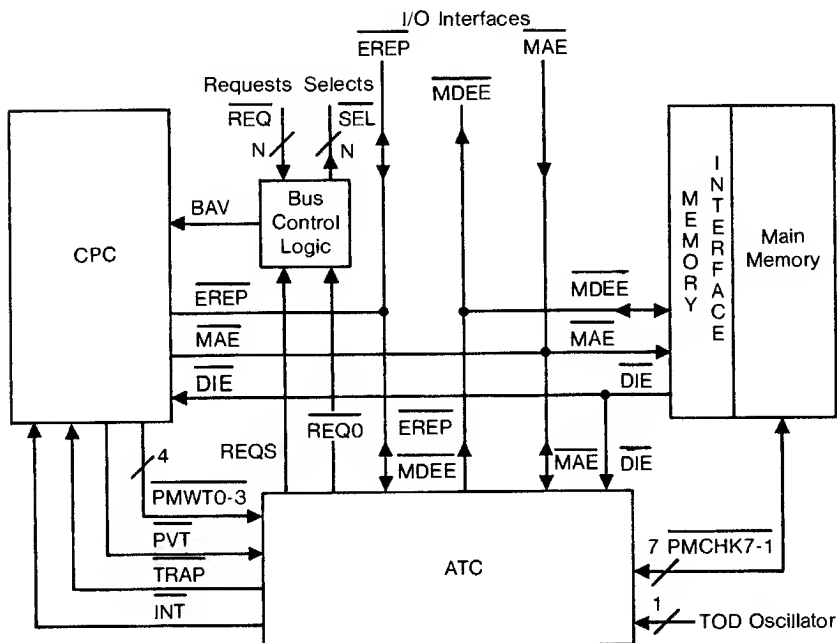
### CONTENTS

General. . . . .	3-2
Logic Conventions. . . . .	3-2
Common PM Bus Signals. . . . .	3-2
External Register Messages. . . . .	3-3
External Register Transfer Signal. . . . .	3-3
External Register Message Transfers. . . . .	3-4
Real Memory Messages. . . . .	3-4
Real Memory Transfer Signals. . . . .	3-4
Real Memory Message Transfers. . . . .	3-6
Virtual Memory and Memory Refresh Messages. . . . .	3-9
Processor PM Bus Transfers. . . . .	3-9
Processor Signals Related to PM Bus Transfers. . . . .	3-9
Processor PM Bus Access. . . . .	3-11
CPC Access to the PM Bus. . . . .	3-11
Special Request PM Bus Access. . . . .	3-11
Double Error Detection. . . . .	3-12
ATC Intervention During Message Transfers. . . . .	3-12
Virtual Memory Operations. . . . .	3-12
Real Memory Operations. . . . .	3-12
ATC Refresh Operations. . . . .	3-14
PM Bus Timing. . . . .	3-14
Virtual Message Transfer Timing. . . . .	3-14
Virtual Memory Fetch Timing. . . . .	3-15
Virtual Memory Full Store Timing. . . . .	3-17
Virtual Memory Partial Store Timing. . . . .	3-17
Real Memory Transfer Timing. . . . .	3-20
CPC Real Memory Fetch Timing. . . . .	3-20
CPC Real Memory Full Store Timing. . . . .	3-22
CPC Real Memory Partial Store Timing. . . . .	3-22
External Register (ERU) Timing. . . . .	3-25
Processor Interrupt Message. . . . .	3-26
Memory Interface. . . . .	3-26

## CHAPTER III PROCESSOR-MEMORY BUS

The Processor-Memory Bus (PM Bus) is the communication path between the NCR/32 Family devices and main memory that is used to transfer addresses, data, and control information. This chapter describes the PM Bus operation including PM Bus message formats, real memory transfers, virtual memory transfers, and external register transfers.

The PM Bus is a 32-bit bidirectional multiplexed bus which utilizes two non-overlapping clocks for each bus cycle. Figure 3-1 shows a typical NCR/32 system configuration including the PM Bus control signals.



Note: Common signals not shown in diagram.

- 1: PM Bus 32-01
- 2: Clocks (X0, X1)
- 3: Reset (PMRST)

GIM304.1

Figure 3-1 Typical PM Bus Configuration

## GENERAL

The PM Bus and its associated control lines consist of a number of uni-directional and bi-directional TTL compatible lines. These lines are available to all devices.

There are three general categories of messages transferred over the PM Bus:

- External Register Unit (ERU) messages
- Real Memory messages
- Virtual Memory messages

The PM Bus operates in two stages. During the first stage a device is selected (granted access to the bus), and during the second stage a transfer takes place. All active devices (devices which can initiate a message transfer) gain access to the bus during the cycle preceding the message transfer by asserting  $\overline{REQ}$ . Each device in the NCR/32 Family, except the CPC, has a Request/Select signal pair associated with it. The ATC, though, does not monitor its select signal since, as the highest priority requestor, it can “take” the PM Bus. The highest priority requestor is granted a one cycle access to the PM Bus by the bus priority logic and, if necessary conditions are met, transfers a message in the subsequent bus cycle. The Bus priority control logic must be implemented external to the NCR/32 Family devices. The CPC does not have a Request/Select signal pair, but rather is granted access to the PM Bus, via BAV, in the absence of any bus requests.

## Logic Conventions

The PM Bus is a negative logic bus. Thus a binary (logical) 1 is represented by a low voltage (0 volts) on the bus, and a binary (logical) 0 by a high voltage (5 volts) on the bus. If a barred term (e.g.,  $\overline{ERE\overline{P}}$ ) is described as being active, activated, or asserted, it is at a low level (0 volts) on the PM Bus. If a barred term is described as being inactive, negated, or not asserted, it is at a high level (5 volts) on the Bus.

If a term is not barred (e.g., BAV) it is active, activated, or asserted at a high level (5 volts) on the PM Bus. Likewise, it is inactive or not asserted at a low level (0 volts) on the Bus.

## Common PM Bus Signals

The signals defined below are available to all PM Bus devices. Those signals relating to a particular message transfer, and signal timing relationships, are described in later sections. All barred signals are active low.

**CLOCK (X0 AND X1)**

These clock lines are driven by the system clock. CLOCK 0 (X0) is the first phase clock of the bus cycle, and CLOCK 1 (X1) is the second. All devices send and receive messages on the bus synchronous to these clocks.

 **$\overline{\text{PMRST}}$  (PM BUS RESET)**

This signal, when active, holds all appropriate logic in a reset state.  $\overline{\text{PMRST}}$  is usually driven by the system power control logic.

 **$\overline{\text{PMBUS32}} - \overline{\text{PMBUS01}}$  (PROCESSOR-MEMORY BUS)**

These bidirectional lines are used to transfer up to 32 bits of information from one device on the PM Bus to another. Address information (a memory address or an ERU address) is transferred from a device that has been granted the PM Bus to a destination device during clock X0. Data information is transferred between the devices during X1. During memory operations between the memory interface and the ATC, data may also be transferred during X0.

 **$\overline{\text{REQ0}} - \overline{\text{REQn}}$  (REQUEST)**

These signals are used for gaining access to the PM Bus through the bus priority logic and are not needed in a system utilizing only the CPC and ATC. They are reserved for use when I/O devices are added to the NCR/32 system.

 **$\overline{\text{SEL0}} - \overline{\text{SELn}}$  (SELECTS)**

These signals are used for gaining access to the PM Bus through the bus priority logic and are not needed in a system utilizing only the CPC and ATC. They are reserved for use when I/O devices are added to the NCR/32 system.

**NOTE:** The Requests and Selects are generalized to REQX and SELX when referenced.

**EXTERNAL REGISTER MESSAGES**

There are up to 96 external register locations available to the CPC (ERU32-ERU127). There are an additional 32 registers (IRU0-IRU31) which occupy the first 32 address locations. These registers, however, are internal to the CPC and therefore are not accessible through the PM Bus. The ERU register locations are implemented external to the CPC and are accessed over the PM Bus. See Chapter IV for the specific allocation of these register locations.

**External Register Transfer Signal**

The following signal is used by those PM Bus devices which receive or transmit External Register messages.



**$\overline{\text{EREP}}$  (EXTERNAL REGISTER ENABLE/PERMIT)**

This signal is generated by the CPC to enable the transfer of an External Register message over the PM Bus. All PM Bus devices that are capable of receiving External Register messages monitor the PM Bus during X0, when  $\overline{\text{EREP}}$  is asserted.

This signal has a second use when I/O devices are connected to the PM Bus. The  $\overline{\text{EREP}}$  signal is monitored during clock X1 by I/O devices which are attempting to transfer an External Register message to the Bus Interrupt Register (BIN) in the ATC (I/O devices send status information to the CPC via the BIN register in the ATC). If  $\overline{\text{EREP}}$  is set active (low) by the ATC during X1, indicating that the BIN is full, then External Register message transfers are not permitted to this register until  $\overline{\text{EREP}}$  is inactive (high) during a subsequent X1.

**External Register Message Transfers**

An External Register transfer is a three-stage operation consisting of (1) bus arbitration leading to device selection, (2) register selection during X0, and (3) data transfer during X1. The CPC preparing to initiate an External Register message transfer formats the information as indicated in Figure 3-2. The 7-bit External Register number field is set to specify the proper destination External Register. The Direction of Transfer bit (PM Bus bit 8) specifies whether the destination External Register is to receive or transmit data during clock X1. If the bit is true ( $\overline{\text{PMBUS08}}$  active, low) the destination External Register receives data from the selected device; if the bit is false ( $\overline{\text{PMBUS08}}$  inactive, high) the destination External Register sends data to the selected device. As an example, if the CPC sends a TOE (Transfer Out External) command to address Hex 2C (the Time-of-Day register in the ATC), the first 7 address lines will be Hex 2C and the 8th bit will be low (logical 1) indicating that data will be transferred from the CPC to the TOD Register.

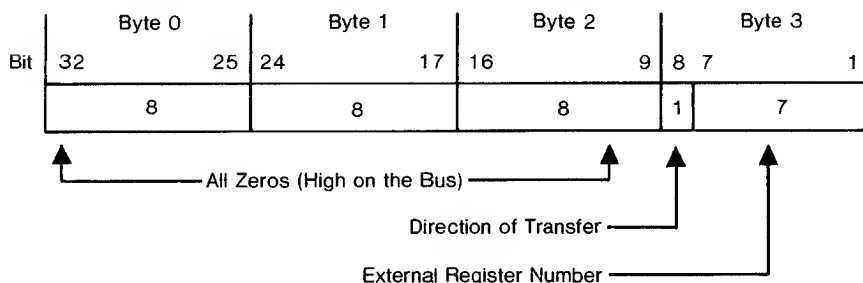
**REAL MEMORY MESSAGES**

All PM Bus devices accessing main memory initiate Real Memory Fetch, Real Memory Full Store, and Real Memory Partial Store operations by transferring Real Memory messages as described in this section. The Refresh message is described in the Memory Interface section of this chapter.

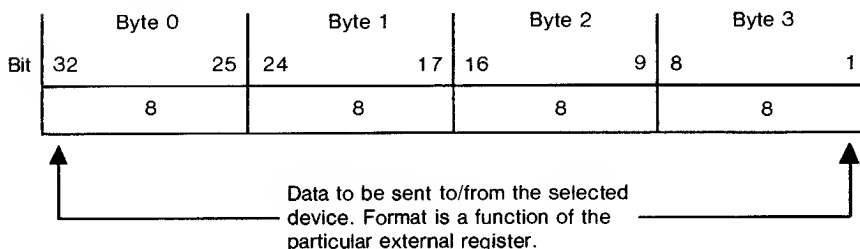
**Real Memory Transfer Signals**

The following signals are used by those PM Bus devices which transfer Real Memory messages.

(a) PM Bus at clock X0 (ERU address to destination device)



(b) PM Bus at clock X1



External Register Messages use the PM Bus during both Clock X0 and X1

GIM3024A

Figure 3-2 External Register Message Format

### MAE (Memory Address Enable)

This signal is generated by the selected device (the active device that has been granted the PM Bus during the previous cycle via REQX and SELX) during X0 to enable the transfer of a Real Memory message over the PM Bus. The memory interface and the ATC monitor the PM Bus during X0 for MAE active (low).

### MDEE (Memory Data Enable/Error)

This signal must be asserted by the memory interface during the X0 clock phase of PM Bus data transfer cycles only in systems utilizing the System Interface Controller (SIC), an NCR/32 Family device currently available. MDEE need not be asserted by the memory interface in systems utilizing only the CPC and ATC. MDEE assertion by the memory interface is therefore not shown in the timing diagrams presented in this chapter.

This signal is also asserted during X1 by the ATC when an uncorrectable (double bit) memory error is detected during a

fetch operation. Although this signal is generated during X1 by the ATC, it is only used at this time by I/O devices and can be ignored when the CPC and ATC are the only devices on the PM Bus.  $\overline{\text{MDEE}}$  is monitored by the PM Bus I/O device that initiated the Real Memory Fetch message.  $\overline{\text{MDEE}}$ , if active during the X1 that the data is available, invalidates the data transferred. This signal is also generated during X1 by the ATC whenever an uncorrectable (double bit) memory error is detected during a Partial Store or a Refresh operation.  $\overline{\text{MDEE}}$  is monitored by the memory interface during X1, and if active, the subsequent Full Word Store which normally occurs during the Partial or Refresh operation is aborted. If the CPC originated the transfer, the ATC informs the CPC of the double bit error through a trap.

Real memory messages potentially use the PM Bus during both clock X0 and X1. The formats differ for Real Fetch, Real Full Store, and Real Partial Store operations. Figures 3-3, 3-4, and 3-5 show the format for these message transfers.

### Real Memory Message Transfers

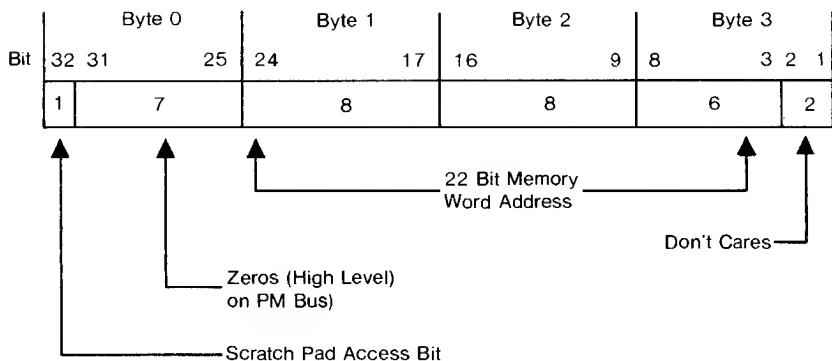
A Real Memory operation consists of a sequence of PM Bus transfers. That sequence involves a device (such as the CPC or ATC) initiating the Real Memory message transfer, the ATC maintaining the integrity of the data, and the memory interface sourcing/receiving the data.

A Real Memory message transfer is in effect a two-bus cycle operation: the first cycle for selection, and the second cycle for message transfer. A PM Bus device preparing to transfer a Real Message formats the information as indicated in Figures 3-3, 3-4, and 3-5. The four Byte Write Enables ( $\overline{\text{PMBUS28-25}}$ ) are appropriately set to indicate into which bytes in the addressed memory location the data will be written (e.g.,  $\overline{\text{PMBUS28}}$  active enables a write by byte 0).

Having been granted bus access, the selected device gates the appropriate X0 message onto the PM Bus and activates the  $\overline{\text{MAE}}$  line during X0. During the following X1 of Memory Full Store or Memory Partial Store operations, the data information to be stored into memory is transferred onto the PM Bus by the selected device.

During Memory Fetch operations, the selected device does not use the PM Bus during the X1 following the Memory Fetch message transfer. The memory interface should assert  $\overline{\text{DIE}}$  during the first X1 clock during which memory is ready, and hold  $\overline{\text{DIE}}$  asserted through the following X0 clock. The memory interface may be required by certain I/O devices on the PM Bus to assert  $\overline{\text{MDEE}}$  during the first X0 clock during which memory is ready. The ATC latches

(a) PM Bus at clock X0 (word address to memory)



(b) PM Bus at Clock X1

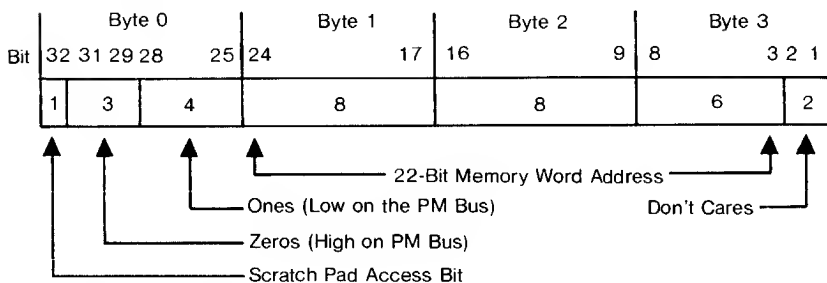
Not Used

Note: The device issuing the Memory Fetch message cannot gate information onto the PM Bus during clock X1.

GIM3025A

Figure 3-3 Real Memory Fetch Message Format

(a) PM Bus at Clock X0



(b) PM Bus at Clock X1

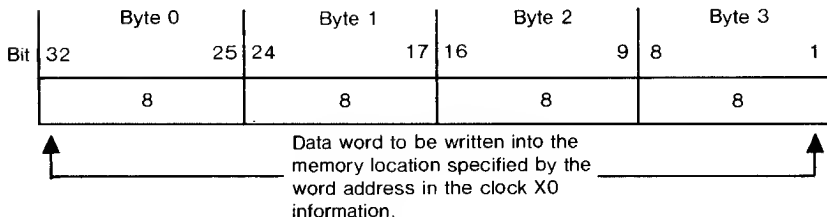
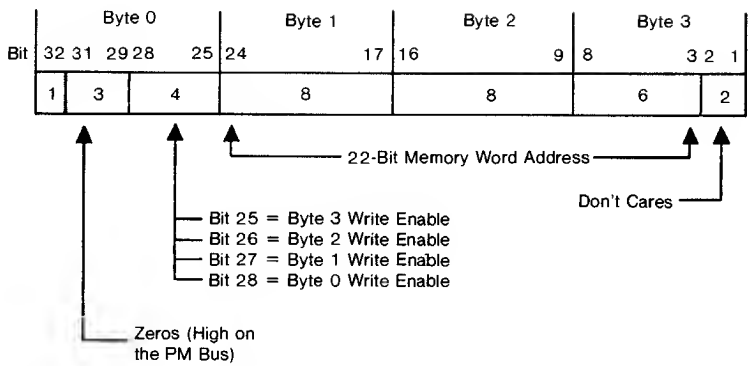


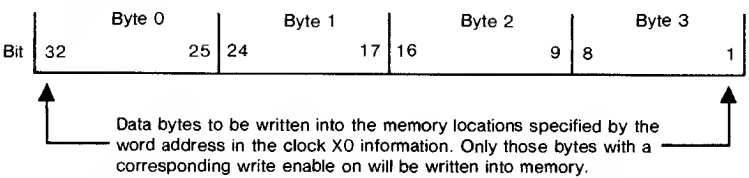
Figure 3-4 Real Memory Full Store Message Format

# PROCESSOR-MEMORY BUS

(a) PM Bus at Clock X0



(b) PM Bus at Clock X1



- Notes:
1. When all Byte Write Enables ( $\overline{\text{PMBUS28-25}}$ ) are on (active low) the memory operation by definition becomes a Memory Full Store.
  2. Bits in the bytes that are not to be written into memory during Partial Stores may be either ones or zeroes.

GIM3027A

Figure 3-5 Real Memory Partial Store (Message Format)

the data during this X0 clock period and checks it for errors. Then, during the following X1, the device reads the information that is on the PM Bus, which at this time is checked/corrected memory data. If an uncorrectable memory error was detected by the ATC during the check/correction phase, then the data read is invalid. The Memory Data Enable/Error ( $\overline{\text{MDEE}}$ ) line is then asserted by the ATC during X1 to indicate the invalidity of the data to the receiving device. The function of this signal during X1 is reserved for use by I/O devices, such as the SIC. In the case of the CPC initiating the transfer, the ATC asserts  $\overline{\text{TRAP}}$  for the CPC. The  $\overline{\text{MDEE}}$  signal is still generated by the ATC but not used by the CPC.

During Memory Fetch and Memory Full Store operations, the device which initiates the message transfer sets bit 32 of the X0 message to the appropriate logic level. If the memory access is referencing the Scratch Pad portion of the main memory via an "all ones" address, then  $\overline{\text{PMBUS32}}$  is set active (low). Otherwise, it is

set inactive (high).

## **VIRTUAL MEMORY AND MEMORY REFRESH MESSAGES**

Figures 3-6 and 3-7 show the Virtual Memory message and Memory Refresh message formats. Virtual Memory accesses are essentially Real Memory accesses preceded by an address translation cycle; Memory Refresh accesses are essentially Real Memory accesses distinguished by the assertion of  $\overline{\text{PMBUS29}}$  in the message (see PM BUS TIMING and MEMORY INTERFACE).

## **PROCESSOR PM BUS TRANSFERS**

The Processor system referred to in this section includes the CPC, ATC, and a memory interface. The transfers described assume that the ATC is used.

The NCR/32-000 processor is capable of performing External Register transfers, Real Memory transfers and Virtual Memory transfers. The CPC initiates all these message transfers through a mechanism different from that of other active PM Bus devices. That mechanism is described in this section. The ATC initiates Real Memory transfers and Memory Refresh transfers through the standard Request/Select protocol.

### **Processor Signals Related to PM Bus Transfers**

The following signals are used by the devices in the Processor subsystem to perform transfers via the PM Bus, including Virtual Memory transfers. All barred signals are active low.

#### **$\overline{\text{BAV}}$ (BUS AVAILABLE)**

This signal is sourced by the Bus Control logic (circuitry external to the NCR/32 Family devices) to the CPC to indicate whether the PM Bus will be available for the CPC in the next bus cycle. If there are no Requests present at the Bus Control logic (all  $\overline{\text{REQX}}$  high), then  $\overline{\text{BAV}}$  will be high and the bus will be available.

#### **$\overline{\text{PVT}}$ (PROCESSOR VIRTUAL TRANSFER)**

This signal is generated by the CPC during clock X0 to enable the transfer of a Virtual Memory message to the ATC over the PM Bus. The ATC begins a Virtual Memory operation when  $\overline{\text{PVT}}$  becomes active. The signal is also generated by the CPC during X1 of all CPC memory transfers via the PM Bus to identify the CPC as being the sender of the message.

#### **$\overline{\text{PMWT0-3}}$ (PROCESSOR-MEMORY WRITE TAGS)**

These lines are asserted by the CPC during X0 when executing Virtual Memory Store transfers to the ATC to indicate which

bytes of the data word (following on the PM Bus during X1) are to be written into memory. These lines are functionally equivalent to the Byte Write Enables (PMBUS28-25) used during Real Message transfers, but are unique lines apart from the 32 PM Bus lines. PMWT0 enables byte 0, etc.

PMCHK7-1 (PROCESSOR-MEMORY CHECK BITS)

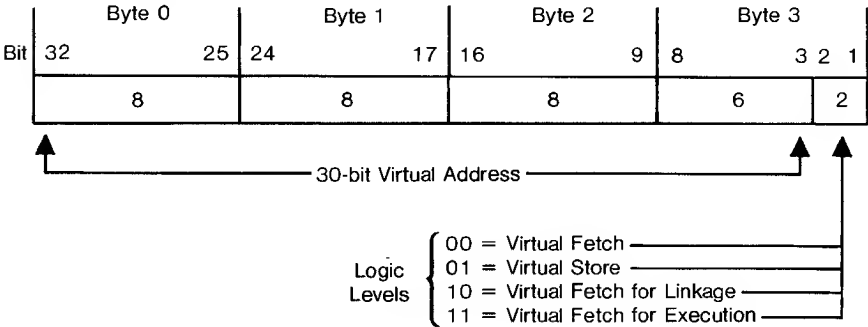
These lines transfer the Check Bits used by the error check/correction logic in the ATC to maintain the integrity of the memory data which is transferred over the PM Bus. During Fetch and Partial Store operations, the Check Bits are sourced by the memory interface to the ATC during the X0 that fetched data is on the PM Bus. During Full Store operations, the Check Bits are sourced by the ATC to the memory interface during the X0 that the stored data is on the PM Bus.

DIE (DATA INPUT ENABLE)

This signal is asserted by the memory interface for use by the ATC and CPC during the X1 time immediately preceding the X0 during which data is asserted onto the PM Bus by the memory interface during Fetch operations. DIE acts as an early indicator that the data is arriving for operations, and that the ATC must use it during the next cycle.

Figures 3-6 and 3-7 show the Virtual Memory message and Memory Refresh message formats.

(a) PM Bus During Clock X0.



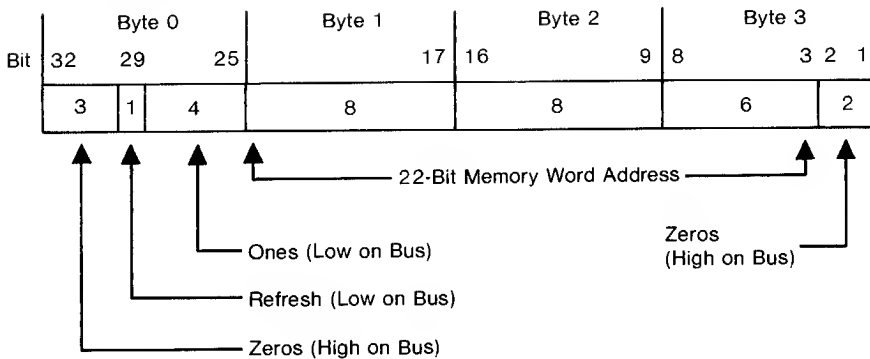
(b) PM Bus During Clock X1.

Data Formats during X1 are identical to Data Formats during X0. Data transferred during X1, however, is received only by the ATC.

GIM3028A

Figure 3-6 Virtual Memory Message Format (PVT Active)

(a) PM Bus at Clock X0



(b) PM Bus at Clock X1

Not Used

Note: The device (ATC) issuing the Refresh message must not gate information onto the PM Bus during X1.

GIM3029A

Figure 3-7 Memory Refresh Message Format ( $\overline{\text{MAE}}$  Active)

### Processor PM Bus Access

The CPC mechanism for gaining PM Bus access and the ATC assertion of the Special Request signal are detailed in this section.

**CPC Access to the PM Bus**—The CPC is granted access to the PM Bus, under normal circumstances, only if there are no requests pending for the bus. All active devices generating a request do so at the beginning of X0. If no requests are present at the Bus Control logic at this time, the BAV signal will be seen active (high) by the CPC. During X1 the CPC will determine from BAV whether the PM Bus is available for use in the next cycle.

Under certain conditions the CPC may require use of the PM Bus unconditionally during a given cycle. In this case the ATC asserts the Special Request signal (REQS) for the Bus Control logic, overriding any requests that might be present from other devices. The Bus Control logic then activates BAV for the CPC to enable the next cycle transfer.

**Special Request PM Bus Access**—The ATC generates the Special Request (REQS) signal to the Bus Control whenever the CPC re-



quires immediate access to the PM Bus. As long as REQS remains active the requests from other devices will be ignored by the Bus Control, and no selects will be issued.

Certain conditions, such as uncorrectable memory errors detected during CPC-initiated Fetches, require CPC intervention prior to allowing other transfers on the PM Bus. In these cases the ATC asserts REQS.

### DOUBLE ERROR DETECTION

The ATC performs error check/correction during memory operations. When the ATC detects a double-bit error in the fetched data during a Real Memory Fetch or a Partial Store operation, it asserts the Memory Data Enable/Error signal (MDEE) during X1. An asserted MDEE during X1 is reserved for use by the SIC. The ATC reports uncorrectable errors to the CPC through a trap.

However, MDEE should be used by the memory interface during partial store and refresh operations to abort writes into memory when a double-bit error is detected by the ATC.

### ATC INTERVENTION DURING MESSAGE TRANSFERS

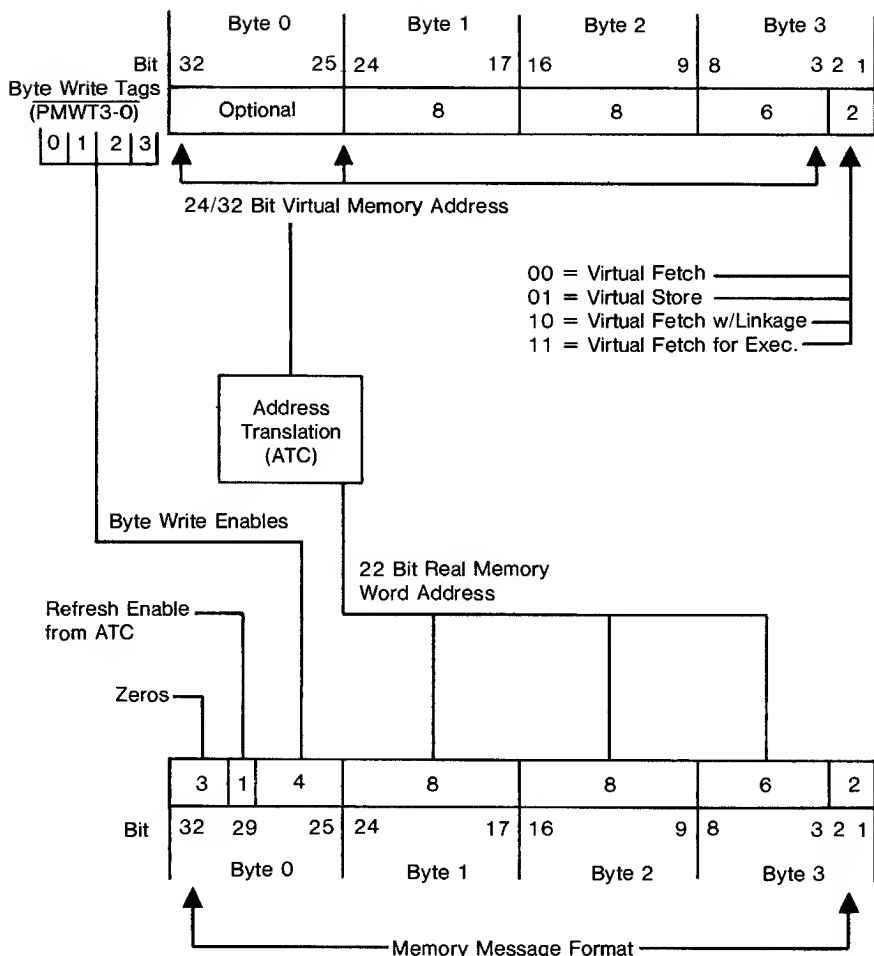
The ATC, when enabled, intervenes in all memory operations. During virtual memory operations the ATC receives a Virtual memory message from the CPC, then sends a Real Memory message to the memory interface. During real memory operations the ATC receives a Real Memory message from a device, then sends data to the memory interface (if a store) or returns data to the initiating device (if a fetch).

### Virtual Memory Operations

The ATC performs the required address translation during virtual memory operations. The ATC translates the virtual address in the Virtual memory message into a real memory address, and sends a Real Memory message to the memory interface. The translation process is performed during the bus cycle that the Virtual Memory message is transferred over the PM Bus. Virtual message translation is shown in Figure 3-8.

### Real Memory Operations

The ATC performs the check bit verification/generation, when enabled, during all (real) memory operations. During Fetch operations the ATC receives the accessed word from the memory interface and verifies the integrity of the data by using Check Bits. The ATC corrects single bit errors, and asserts MDEE during X1 if it detects a double bit error. If the ATC detects a double bit error during a CPC-initiated transfer, it will also force a CPC trap. In all cases, the ATC



GIM3031A

Figure 3-8 Virtual Address Translation (Virtual Memory Message conversion to real Memory Message Format)

passes data to the initiating device. During Full Store operations, the ATC receives the data word to be stored from the initiating device, and then generates the appropriate Check Bits for that data pattern. The ATC then transfers the data and Check Bits to the memory interface.

During Partial Store operations the ATC receives the data word containing the bytes to be stored from the initiating device. The ATC then receives from the memory interface the current contents of the memory location where the data bytes are to be stored. This data word is checked and single bit errors corrected. The data bytes that are to be written into memory are then substituted into the checked/corrected

data word, and a new set of Check Bits are generated by the ATC for the resulting word. A double-bit error detected during the check cycle will force  $\overline{\text{MDEE}}$  active during X1, inhibiting the normal write into memory by the memory interface. A subsequent Fetch of the same memory word will again result in a double-bit error.

### ATC Refresh Operations

The TOD (Time-of-Day) Counter in the ATC is used to generate the timing and the memory addresses used for refresh operations. The refresh message is transferred to the memory interface at intervals determined by the refresh rate of the Dynamic RAM used in the memory array. The refresh message enables the memory interface to refresh all memory cells within one row address for all banks of memory, and to place the data word specified by the entire refresh address onto the PM Bus. By this scheme the memory is constantly refreshed and periodically verified at every memory location.

The ATC receives and checks the memory word, single-bit errors are corrected and reported with the assertion of  $\overline{\text{MEMERR}}$ , double-bit errors reported. The Check Bits are regenerated, and if a double error was detected,  $\overline{\text{MDEE}}$  is activated during X1. The ATC then transfers the data and Check Bits to the memory interface where a full word is stored (unless  $\overline{\text{MDEE}}$  was activated due to a multiple bit error).

The timing of a Refresh operation is identical to that of a Real Partial Store. However, the Refresh message has all Byte Write Enables asserted. The Refresh Enable bit in the Refresh message (Refresh Enable =  $\overline{\text{PMBUS29}}$ , active) forces the memory interface to perform a Fetch followed by a Full Store as is required for a Partial Store. Table 3-1 presents a summary of message control signals.

### PM BUS TIMING

This section shows and explains PM Bus timing for each type of PM Bus message transfer. The ATC and CPC technical information publications (data sheets) should be referred to for precise timing parameters.

#### Virtual Message Transfer Timing

All Virtual Memory operations are 1 cycle longer than their real memory counterparts. The additional cycle is required for the translation of the virtual address into a real memory address. Virtual Fetches are nominally 3 cycles as are virtual full stores. Virtual Partial Stores are 4 cycles. All memory operations, virtual or real, may require additional cycles when slow memory devices are used. Table 3-2 summarizes the number of bus cycles required to complete each of the virtual memory operations.

Controls (Unprimed)					Operation	Source		Destination		
$\overline{\text{MAE}}$	$\overline{\text{EREP}}$	$\overline{\text{PVT}}$	Byte Enable	PM Bus 02, 01		CPC	ATC	MEM	CPC	ATC
X			All Off	0, 0	Real Memory Fetch*	X	X	X		X
X			All On	0, 0	Real Memory Word Store	X	X	X		X
X			1, 2, or 3	0, 0	Real Memory Part. Store	X		X		X
		X	All Off	0, 0	Trans. Virt. w/Read Check	X				X
		X	1, 2, 3, or 4 On	0, 1	Trans. Virt. w/Write Check	X				X
		X	All Off	1, 0	Trans. Virt. w/Link Check	X				X
		X	All Off	1, 1	Trans. Virt. w/Exec Check	X				X
	X		Not Used	**	Trans. Out/In to Ext. Reg.	X				X
X			All On	0, 0	Refresh		X	X		

\*Data is returned to source device, and to ATC where Check/Correction is performed

\*\* $\overline{\text{PMBUS08}}$  active indicates a transfer out from the source.  $\overline{\text{PMBUS08}}$  inactive indicates a transfer in to the source. PM Bus bits 02 and 01 are used for the ERU address

GIMTE3030A

Table 3-1 Summary of Message Controls

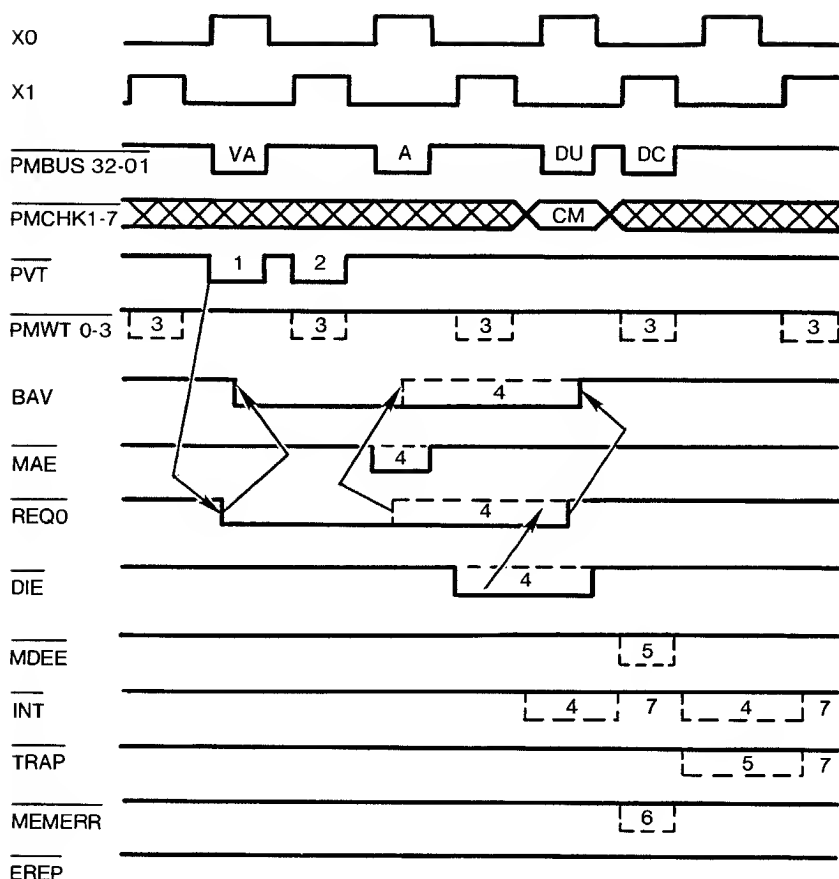
**Virtual Memory Fetch Timing**—Figure 3-9 shows the timing relationships of the control signals used during a Virtual Memory Fetch operation.

The CPC must first be granted access to the PM Bus (BAV asserted). The CPC initiates the Virtual Memory Fetch during X0 by asserting  $\overline{\text{PVT}}$ . At this time the CPC asserts the virtual address on the PM Bus, and forces all  $\overline{\text{PMWT0-3}}$  inactive to define the operation as a Fetch.

If the address translation is successful, then the ATC asserts the real memory address on the PM Bus ( $\overline{\text{PMBUS24-03}}$ ) during the following X0 and asserts  $\overline{\text{MAE}}$ , forcing the memory interface to perform a fetch operation.

If the address translation is unsuccessful, the ATC does not assert  $\overline{\text{MAE}}$  and instead interrupts the CPC. If the ATC cannot correct a data error in the fetched data, the ATC will trap the CPC.

## PROCESSOR-MEMORY BUS



VA = Virtual Address from CPC

A = Real Address from ATC

CM = Check Bits from Memory Interface

DU = Data unchecked from Memory Interface

DC = Data checked from ATC

1. Asserted by CPC to initiate virtual transfer.
2. Asserted by CPC to indicate CPC-initiated memory transfer.
3. Asserted/read by CPC for  $\overline{\text{BKPTWE}}$ ,  $\overline{\text{BKPTe}}$ ,  $\overline{\text{VPBSY}}$ , and  $\overline{\text{BCT}}$  functions.
4. If address translation is unsuccessful, ATC does not assert  $\overline{\text{MAE}}$ , aborting the memory operation. ATC asserts  $\overline{\text{INT}}$  if enabled.
5. ATC asserts  $\overline{\text{MDEE}}$  and  $\overline{\text{TRAP}}$  if it detects an uncorrectable error.
6. ATC asserts  $\overline{\text{MEMERR}}$  if it detects either a correctable (single bit) or uncorrectable (multiple bit) data error.
7. ATC negates  $\overline{\text{INT}}$  and  $\overline{\text{TRAP}}$  during X1.

GIM3042

Figure 3-9 Virtual Memory Fetch Timing

$\overline{\text{MDEE}}$  is asserted by the memory interface for use by I/O devices during X0 to indicate that a memory operation is about to be completed (see  $\overline{\text{MDEE}}$  description, page 3-5).

$\overline{\text{MDEE}}$  is asserted by the ATC during X1 if it detects an uncorrectable data error, forcing an abort of write into memory.  $\overline{\text{MEMERR}}$  is asserted by the ATC during X1 if it detects either a correctable (single bit) or an uncorrectable (multiple bit) data error. The states of  $\overline{\text{MDEE}}$  and  $\overline{\text{MEMERR}}$  at this time allow the system to determine whether (1) there was no data error, (2) there was a corrected data error, or (3) there was an uncorrectable data error.

**Virtual Memory Full Store Timing**—The diagram in Figure 3-10 shows the timing relationships of the control signals used during Virtual Full Store transfers on the PM Bus. The timing diagram assumes that the CPC has been granted access to the bus and is initiating the transfer. The ATC performs an address translation on the virtual address received in the message and issues a Real Memory message. Table 3-2 summarizes the number of bus cycles required to complete each of the virtual memory operations.

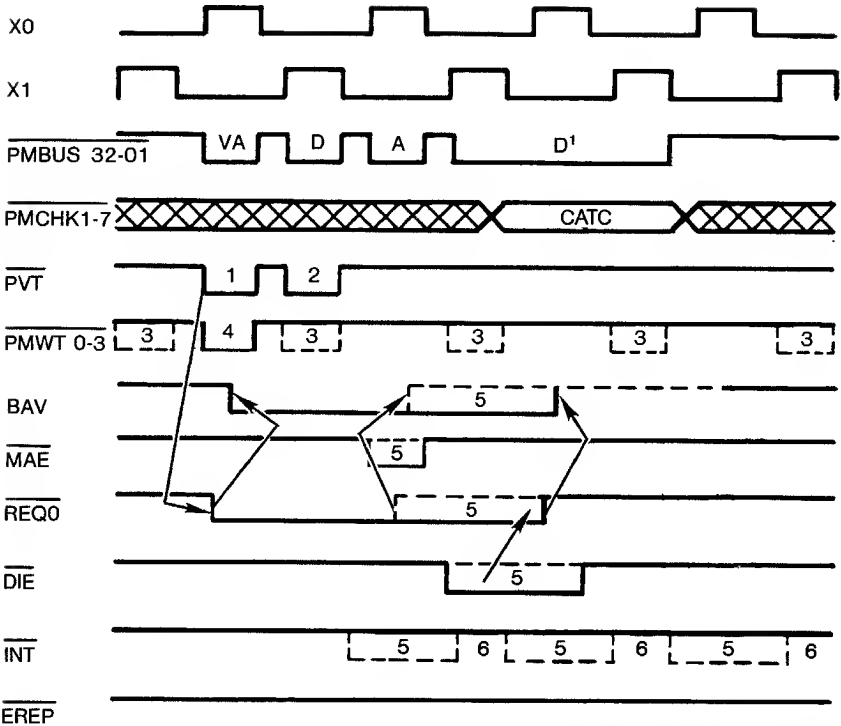
The CPC transfers a 30-bit virtual address on to  $\overline{\text{PMBUS32-03}}$ , asserts the write enable signals ( $\overline{\text{PMWT0-3}}$ ), and asserts the Processor Virtual Transfer signal ( $\overline{\text{PVT}}$ ) during X0. The Write Enable signals are asserted separately on  $\overline{\text{PMWT0-3}}$ , since the address is expanded to 30 bits. After address translation to a 22-bit word address, the Write Enable Signal functions are transferred to  $\overline{\text{PMBUS28-25}}$ .

If the address translation is successful, then the ATC asserts  $\overline{\text{MAE}}$  the following X0, allowing the memory interface to perform the store operation. If the translation is unsuccessful, the ATC generates an interrupt, does not assert  $\overline{\text{MAE}}$ , and releases the bus (deactivates  $\overline{\text{REQ0}}$ ).

During slow memory operations the memory interface does not assert  $\overline{\text{DIE}}$  until one cycle (or more) later than usual. This suspends the ATC in the proper state to either remain transmitting data (if a store), or to keep waiting for data (if a fetch).

**Virtual Memory Partial Store Timing**—Figure 3-11 shows the timing relationships of the control signals used during a Virtual Memory Partial Store operation.

The CPC first gets access to the PM Bus, then initiates the virtual operation by asserting  $\overline{\text{PVT}}$ . At this time (X0) the CPC asserts the virtual address on the PM Bus, and asserts one, two, or three of the write enables ( $\overline{\text{PWMT0-3}}$ ) to define the operation as a partial store. The CPC asserts the data to be written on the PM Bus the following X1.



VA = Virtual Address from CPC

D = Data from CPC

CATC = Check Bits from ATC

A = Real Address from ATC

D1 = Data from ATC

1. Asserted by CPC to initiate virtual transfer.
2. Asserted by CPC to indicate CPC-initiated memory transfer.
3. Asserted/read by CPC for BKPTWE, BKPTE, VPBSY, and BCT functions.
4. All PMWT strobes asserted.
5. If address translation is unsuccessful, ATC does not assert MAE, aborting the memory operation. ATC asserts INT if enabled.
6. ATC negates INT during X1.

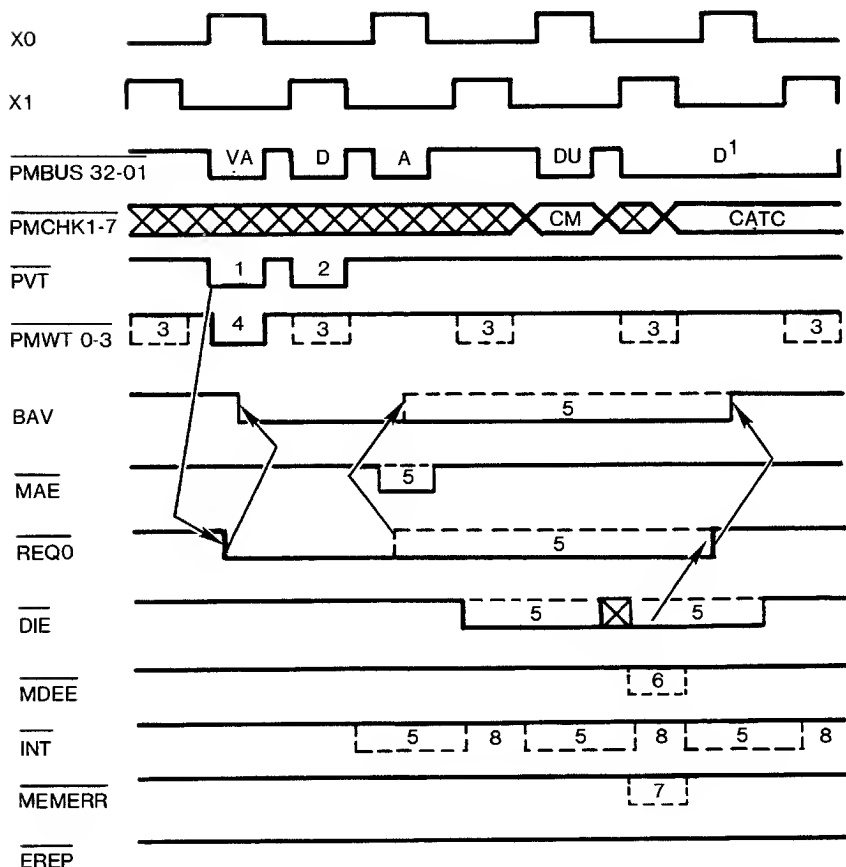
Figure 3-10 Virtual Memory Full Store Timing

Virtual Operation	Virtual Transfer (Cycles)	Real Transfer (Cycles)	Total Cycles
Virtual Memory Fetch	1	2	3
Virtual Memory Full Store	1	2	3
Virtual Memory Part. Store	1	3	4

NOTE: The cycle counts are minimum, assuming no memory wait states.

GIMTE3032A

Table 3-2 Virtual Memory Operation Cycle Counts



VA = Virtual Address from CPC

D = Data from CPC

A = Real Address from ATC

DU = Data Unchecked from Memory Interface

CM = Check Bits from Memory Interface

D<sup>1</sup> = Data from ATC

CATC = Check Bits from ATC

1. Asserted by CPC to initiate virtual transfer.
2. Asserted by CPC to indicate CPC-initiated memory transfer.
3. Asserted/read by CPC for BKPTWE, BKPTE, VPBSY, and BCT functions.
4. One, two, or three PMWT strobes asserted.
5. If address translation is unsuccessful, ATC does not assert  $\overline{\text{MAE}}$ , aborting the memory operation. ATC asserts interrupt if enabled.
6. ATC asserts  $\overline{\text{MDEE}}$  if it detects an uncorrectable data error.
7. ATC asserts  $\overline{\text{MEMERR}}$  if it detects either a correctable (single bit) or uncorrectable (multiple bit) data error.
8. ATC negates  $\overline{\text{INT}}$  during X1.

GIM3044

Figure 3-11 Virtual Memory Partial Store Timing



The ATC performs a virtual address translation and, if the translation is successful, asserts  $\overline{MAE}$  the following X0 and asserts a real (translated) address on the PM Bus. If the translation is unsuccessful, the ATC does not assert  $\overline{MAE}$  and instead interrupts the CPC.

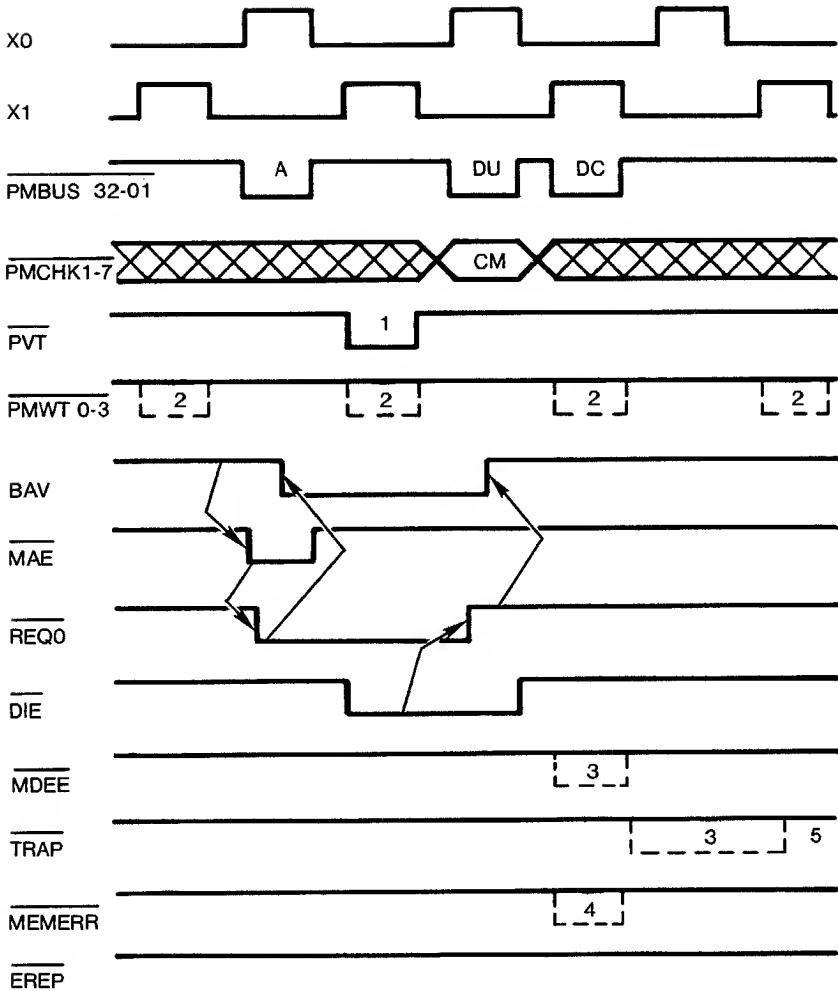
The ATC reads the addressed data word (DU) the following X0, performs error check/correction on the data word, substitutes the new byte(s) into the data word, and asserts the new data word (D1) onto the PMBUS the following X1. The ATC, after internal logic delay, then asserts the data check bits.

The ATC asserts  $\overline{MDEE}$  during X1 if it has detected an uncorrectable data error to alert the memory interface to abort the memory operation, and asserts  $\overline{MEMERR}$  if it has detected any memory error.

### Real Memory Transfer Timing

Figures 3-12, 3-13, and 3-14 show the timing relationship of the control signals used during Real Memory message transfers on the PM Bus. The timing diagrams assume that the requesting device has been granted access to the bus and is initiating the transfer. The ATC performs a check/correct on all data from the memory interface, and generates check bits for all data stored to memory. An insert is performed during Partial store operations for those bytes to be written into memory. Table 3-3 summarizes the number of bus cycles required to complete each of the real memory operations.

**CPC Real Memory Fetch Timing**—Figure 3-12 shows the timing for a Real Memory Fetch by the CPC. The 2-cycle timing is the minimum number of cycles during which a memory fetch can be performed. An asserted BAV gives the CPC access to the PM Bus. The 22-bit word address is asserted on  $\overline{PMBUS24-03}$  by the CPC, the Write Enable Signals are deactivated by the CPC ( $\overline{PMBUS28-25}$  are high), and the Memory Address Enable ( $\overline{MAE}$ ) is activated during X0 by the CPC.  $\overline{MAE}$  forces the ATC to secure the bus through the remainder of the Memory Fetch via the highest priority request ( $\overline{REQ0}$ ). The memory interface activates  $\overline{DIE}$  during the X1 preceding the transfer of the memory data to advance the ATC state sequencer. The memory interface activates  $\overline{MDEE}$  during the subsequent X0 as an enable for I/O devices. The unchecked data (DU) on the PM Bus during X0 is checked and a single-bit correction performed, if required, by the ATC. The checked/corrected data (DC) is asserted on the bus by the ATC during the following X1 and received by the CPC. A multiple-bit error detected by the ATC will activate  $\overline{MDEE}$  during X1. The ATC state transition forced by  $\overline{DIE}$



A = Real Address from CPC

DU = Data Unchecked from Memory Interface

DC = Data Checked from ATC

CM = Check Bits from Memory Interface

1. Asserted by CPC to indicate CPC-initiated memory transfer.
2. Asserted/read by CPC for BKPTWE, BKPTRE, VPBSY, and BCT functions.
3. ATC asserts MDEE if it detects an uncorrectable data error; asserts TRAP if the memory cycle was CPC initiated.
4. ATC asserts MEMERR if it detects either a correctable (single bit) or uncorrectable (multiple bit) data error.
5. ATC negates TRAP during X1.

Figure 3-12 CPC Real Memory Fetch Timing

Real Memory Operation	Read Access (Cycles)	Check/Correct (Cycles)	Insert/Gen. (Cycles)	Write (Cycle)	Total Cycles
Fetch	1	1	—	—	2
Full Store	—	—	1	1	2
Part. Store	1	1/2	1/2	1	3
Refresh *	1	1/2	1/2	1	3

\*Refresh timing is identical to the Partial Store timing. A refresh operation performs a "zero" insert, that is, all fetched bytes are returned to memory.

GIMTE3033A

Table 3-3 Real Memory Operation Cycle Counts

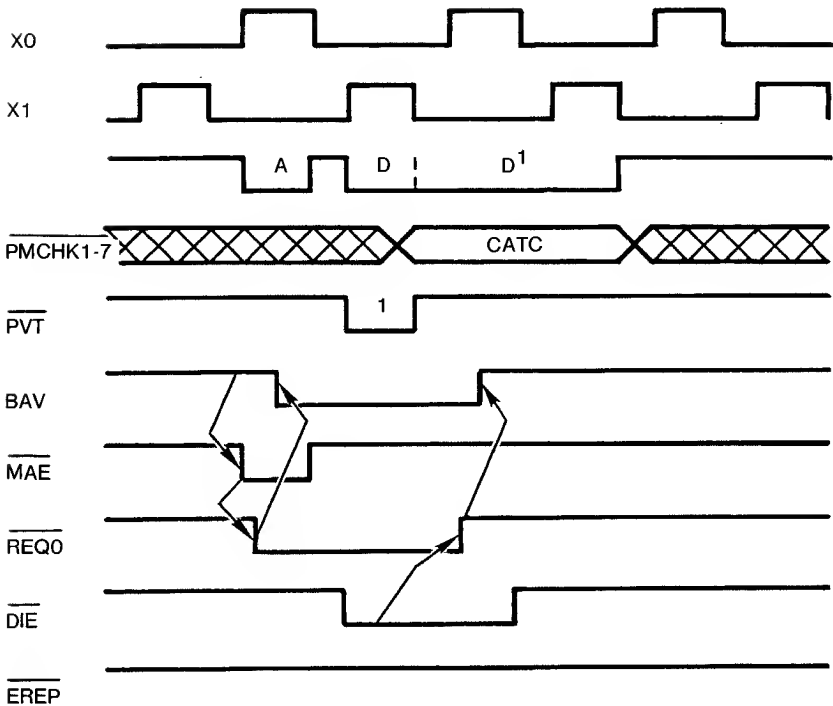
deactivates  $\overline{\text{REQ0}}$  and releases the PM Bus.

I/O device Real Memory Fetch is similar. However, the I/O device uses  $\overline{\text{REQX/SELX}}$  handshaking to secure the Bus, and  $\overline{\text{PVT}}$  is not asserted.

**CPC Real Memory Full Store Timing**—Figure 3-13 shows the timing for a CPC Real Memory Full Store Operation. The 2-cycle timing is the minimum number of cycles during which a memory store can be performed. The CPC initiates the store only after the availability of the bus has been guaranteed ( $\text{BAV}$  high) during the previous cycle. The 22-bit word address is asserted onto  $\text{PMBUS24-03}$ , the Write Enable Signals are all activated ( $\text{PMBUS28-25}$  are low) and the Memory Address Enable ( $\overline{\text{MAE}}$ ) is activated during  $\text{X0}$ . The data to be written into memory is asserted onto the  $\text{PMBUS32-01}$  lines during the following  $\text{X1}$ . Asserted  $\overline{\text{MAE}}$  and the decode of the Memory Write Enables force the ATC to begin generating a 7-bit ECC code for the write data, and force the memory interface to begin the memory cycle. The ATC holds the data on the bus during the following  $\text{X0}$ , and transfers the check bits on the PMCHK Bus ( $\text{PMCHK7-1}$ ) to the memory interface. The memory interface activates the  $\overline{\text{DIE}}$  during the  $\text{X1}$  preceding the transfer of the data from the ATC.

I/O device full store is similar. However, the I/O device uses  $\overline{\text{REQX/SELX}}$  handshaking to secure the Bus, and  $\overline{\text{PVT}}$  is not asserted.

**CPC Real Memory Partial Store Timing**—Figure 3-14 shows the timing for a CPC Real Memory Partial Store operation. The 3-cycle timing is the minimum number of cycles during which a Partial Word Store (less than four bytes) can be performed. The operation is



A = Real Address from CPC

D<sup>1</sup> = Data from ATC

D = Data from CPC

CATC = Check Bits from ATC

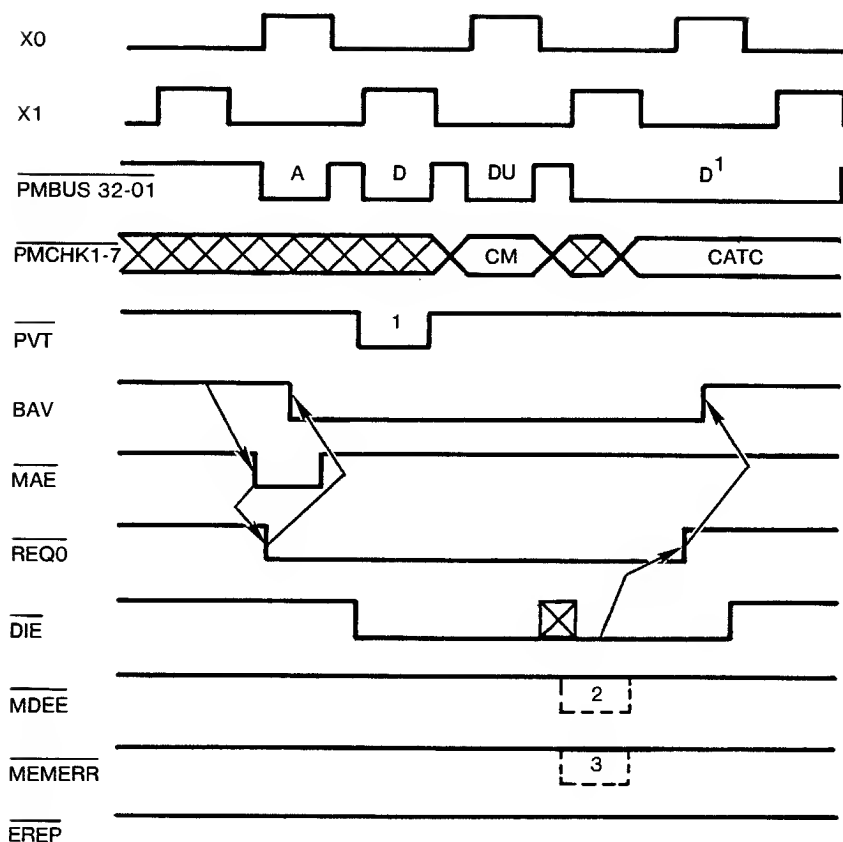
1. Asserted by CPC to indicate CPC-initiated memory transfer

GIM3046

Figure 3-13 CPC Real Memory Full Store Timing

a Read-Modify-Write function in order to generate the proper ECC code for the combination of modified and unmodified data bytes in the memory word. The CPC, having secured the bus, asserts the 22-bit word address onto the bus (PMBUS24-03), activates the appropriate Write Enable Signals (PMBUS28-25) corresponding to the bytes to be written, and activates  $\overline{\text{MAE}}$  during X0. The data is asserted onto the bus by the CPC during the following X1. The memory interface recognizes the Partial Store, transfers the data word at the addressed location during the next X0, and asserts it (and the ECC bits) onto the busses. The ATC receives the data and checks and, if required, corrects the data as during a normal fetch at this time. During the next X1 the ATC substitutes the bytes to be written into memory for those bytes which had been fetched from memory and

# PROCESSOR-MEMORY BUS



A = Real Address from CPC

D = Data from CPC

CM = Check Bits from Memory Interface

DU = Data Unchecked from Memory Interface

D¹ = Data from ATC

CATC = Check Bits from ATC

1. Asserted by CPC to indicate CPC-initiated memory transfer
2. ATC asserts MDEE if it detects an uncorrectable data error
3. ATC asserts MEMERR if it detects either a correctable (single bit) or uncorrectable (multiple bit) data error

GIM3045

Figure 3-14 CPC Real Memory Partial Store Timing

are to be replaced. The resulting data word has a new ECC pattern generated for it, and is transferred to the memory interface during the following X0.

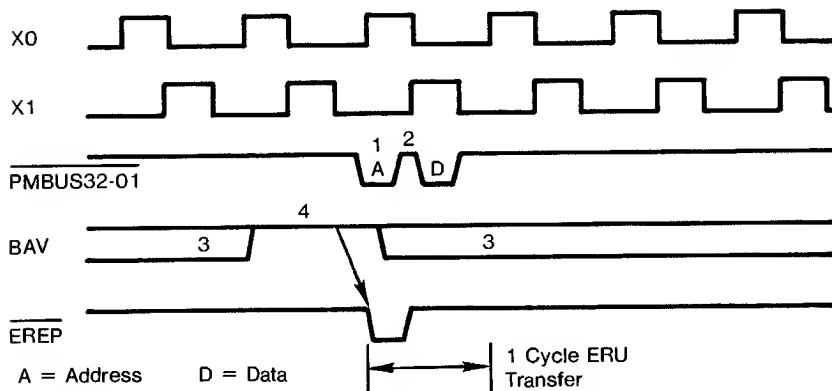
The memory interface activates  $\overline{\text{DIE}}$  during the X1 preceding the data transfer from memory to the ATC, and during the following X1 to acknowledge that memory is ready for the data from the ATC.

I/O device Real Memory Partial Store timing is similar. However, the I/O device uses  $\overline{\text{REQX/SELX}}$  handshaking to secure the Bus, and PVT is not asserted.

The memory interface recognizes a refresh operation when the Refresh Enable bit is asserted in the real message ( $\overline{\text{PMBUS29}}$ ).

The ATC generates a refresh message with the same timing as a Real Partial Store message. The difference is that after the data word is read from memory, no byte substitution is made. The entire word that is read is rewritten back into memory after the ECC code is checked and any single-bit errors corrected. Each refresh message both stimulates the memory interface to perform a dynamic RAM row refresh, and accesses an individual memory word for error check/correction (scrubbing).

**External Register (ERU) Timing**—Figure 3-15 shows the timing for both 1-cycle and 2-cycle ERU operations.



1.  $\overline{\text{PMBUS07-01}}$  contain the 7-bit ERU address and  $\overline{\text{PMBUS08}}$  contains the direction of transfer.
2. PMBUS pulled-up (pre-charged).
3. BAV may be low (bus not available) due to other bus activity such as refresh.
4. Bus is now available.

GIM3038A

Figure 3-15 External Register Message Timing

**1-CYCLE ERU REFERENCE:** The CPC gains access to the bus to perform an External Register Unit (ERU) reference only in

the absence of requests from any other devices on the bus (BAV high). The CPC asserts the 7-bit ERU address onto  $\overline{\text{PMBUS07-01}}$ , activates  $\overline{\text{PMBUS08}}$  (low) for a transfer out from the CPC or leaves inactive  $\overline{\text{PMBUS08}}$  (high) for a transfer into the CPC, and activates the ERU enable signal ( $\overline{\text{EREP}}$ ) during X0. If a Transfer Out, the destination ERU receives data during X1. If a Transfer In, the destination ERU transmits data during X1.

**2-CYCLE ERU REFERENCE:** Certain ERU transfers to the ATC require that the CPC be granted the bus for two consecutive cycles. Since the CPC only gains access to the bus in the absence of requests from other devices, the second cycle must be guaranteed by the ATC. The Special Request (REQS) is activated by the ATC during X0 when  $\overline{\text{EREP}}$  has been sourced by the CPC and a 2-cycle ERU address has been asserted on the bus. REQS should force BAV active for the CPC despite the presence of any requests ( $\overline{\text{REQX}}$ ). REQS should also disable the generation of any selects ( $\overline{\text{SELX}}$ ).

The second cycle is required either for the CPC to perform another ERU reference tightly coupled to the first, or to allow the ATC to complete the operation dictated by the first cycle ERU reference.

## PROCESSOR INTERRUPT MESSAGE

A Processor Interrupt message is an External Register transfer to the Bus Interrupt (BIN) Register in the ATC from another device tied to the PM Bus. The BIN Register is the input path for devices (e.g., I/O devices) other than the ATC to interrupt the CPC. Thus if an I/O device wants to interrupt the CPC, it sends a message to the ATC BIN Register, and subsequently the ATC will interrupt the CPC.

The ATC uses  $\overline{\text{EREP}}$  during clock X1 to control the loading of the BIN Register. The BIN Register is unloaded on an interrupt priority basis by the CPC, and therefore a new transfer into the BIN is not allowed until an old message is read out. The ATC will continue to assert  $\overline{\text{EREP}}$  during all X1 times until the BIN Register is transferred into the CPC. The format of the interrupt message (X1 information in the External Register transfer) is a function of the particular device which sources the message.

## MEMORY INTERFACE

The following describes a typical memory interface connected to the NCR/32 System. The memory interface converts the standard PM Bus line voltage levels to the voltage levels required by the particular memory component used in the memory array. The memory interface is idle until  $\overline{\text{MAE}}$  is asserted during clock X0 by a selected

device. The memory interface, upon detecting  $\overline{\text{MAE}}$  active, generates the row address strobe (RAS) which clocks the row address into the Dynamic RAM chips. The row address is available via the memory interface address multiplexer from the lower portion of the memory word address (PMBUS10-03) that is on the PM Bus during X0. The size of the multiplexer is a function of the density of the memory components in the array. The assumed chip size in this description is a 64k x 1 RAM. The memory interface decodes  $\overline{\text{PMBUS28-25}}$ ,  $\overline{\text{PMBUS29}}$ , and  $\overline{\text{PMBUS32}}$  during the X0 clock to determine the type of memory operation to be performed.

During a Fetch ( $\overline{\text{PMBUS28-25}}$  are all high) the memory interface generates the column address strobe (CAS) as a delayed function of RAS. The CAS signal clocks the column address, which is available via the multiplexer from the upper portion of the memory word address (PMBUS24-11), into the Dynamic RAMs. The accessed word is then asserted onto the PM Bus, and RAS and CAS are deactivated appropriately by the memory interface.

During a Full Store ( $\overline{\text{PMBUS28-25}}$  are all low) the memory interface generates the CAS as a delayed function of RAS. The column address is strobed into the RAM, and the data word on the PM Bus from the ATC is written into the addressed location during the X0 clock period.

During a Partial Store ( $\overline{\text{PMBUS28-25}}$  equals some low) the memory interface generates CAS as a delay of RAS as in the Fetch case, strobes in the column address, and asserts the accessed word onto the PM Bus. CAS is then deactivated by the memory interface while RAS remains active. Then, after an appropriate delay, the memory interface generates CAS again, and the data word on the PM Bus from the ATC is written into the addressed location during the X0 in which  $\overline{\text{MDEE}}$  is asserted.

During a refresh operation ( $\overline{\text{PMBUS28-25}}$  are low,  $\overline{\text{PMBUS29}}$  is low) the memory interface performs the identical operations as during a Partial Store. In addition,  $\overline{\text{PMBUS29}}$  active forces RAS to be asserted for all the banks of RAM chips in memory, not just the one selected bank as is the case during a normal Partial Store (where  $\overline{\text{PMBUS29}}$  is inactive).

During most memory operations,  $\overline{\text{PMBUS24-19}}$  of the word address should be compared against the maximum allowable memory address for the main memory configuration. If the word address is greater than the maximum allowable address, an LGM (Larger than Memory) condition occurs and the memory interface should subsequently present a Check Bit pattern to the ATC that results in an uncorrectable error if the operation was a Fetch. If the operation was a Partial or Full Word Store, then an LGM condition should



## PROCESSOR-MEMORY BUS

force the memory interface to abort the write into the addressed memory location. During refresh operations ( $\overline{\text{PMBUS29}}$  active) or accesses to the Scratch Pad ( $\overline{\text{PMBUS32}}$  active) portion of the memory, however, the LGM should be bypassed.

## CHAPTER IV CENTRAL PROCESSOR CHIP (CPC)

### CONTENTS

CPC Overview . . . . .	4-1
SIGNAL DESCRIPTION . . . . .	4-1
DATA ORGANIZATION IN MEMORY . . . . .	4-7
ISU Memory . . . . .	4-7
Main Memory . . . . .	4-8
THREE-STAGE PIPELINE . . . . .	4-9
ARITHMETIC LOGIC UNIT (ALU) . . . . .	4-9
Arithmetic Operations . . . . .	4-10
Special Decimal ALU Logic . . . . .	4-10
REGISTER DESCRIPTION . . . . .	4-11
Register Storage Unit (RSU) . . . . .	4-11
External Registers (ERUs) . . . . .	4-13
Internal Register Unit (IRU) . . . . .	4-14
Jump Registers (IRU0-IRU7) . . . . .	4-14
Restore Field (IRU8) . . . . .	4-16
State Register (IRU9) . . . . .	4-17
Bits 1-5 — Field Array Bits . . . . .	4-18
Bits 7-10 — Byte Write Tag Bits . . . . .	4-18
Bits 11-14 — RSU Address Bits . . . . .	4-19
Bits 15-16 — Protection Check Code Bits . . . . .	4-19
Indicator Array (IRU16) . . . . .	4-20
Virtual Indicators (IRU17) . . . . .	4-22
Tally Register (IRU18) . . . . .	4-22
Operand Pointers (IRU24) . . . . .	4-23
Stack Pointer (IRU26) . . . . .	4-24
Setup Registers . . . . .	4-24
Setup Register #1 (IRU19) . . . . .	4-24
Setup Register #2 (IRU20) . . . . .	4-25
Setup Register #3 (IRU-10) . . . . .	4-25
Setup Register #4 (IRU11) . . . . .	4-25

## CHAPTER IV CENTRAL PROCESSOR CHIP (CPC)

### CONTENTS (CONTINUED)

Setup Register #5 (IRU18) .....	4-25
Control Array #1 (IRU27) .....	4-26
MARS6 Write Tags (IRU28) .....	4-27
WRITE TAG OPERATION .....	4-28
SCRATCH PAD ACCESS .....	4-28
Access Via the Operand Pointers .....	4-29
Access Via the Stack Pointer .....	4-30
Scratch Pad Access Via ERU Registers .....	4-31
Scratch Pad Access Via FL and SL .....	4-31
INTERRUPTS/TRAPS .....	4-33
Interrupt/Trap Recognition .....	4-33
Interrupt/Trap Servicing .....	4-33
Saving the Machine State .....	4-34
Restoring From Interrupts/Traps .....	4-34
PROCESSOR RESET .....	4-35
PM BUS ACCESS .....	4-35
SETUP ASSIST .....	4-36
Setup Register Applications .....	4-36
Setup Register #1 Application .....	4-36
Setup Register #2 Application .....	4-37
Setup Register #3 Application .....	4-38
Setup Register #4 Application .....	4-38
Setup Register #5 Application .....	4-39
Scratch Pad Virtual Machine Operation .....	4-44
Operand Pointer #1 .....	4-44
Operand Pointer #2 .....	4-44
Stack Pointer .....	4-46
Map Indicator Logic .....	4-46

**CHAPTER IV  
CENTRAL PROCESSOR CHIP (CPC)**

**CONTENTS (CONTINUED)**

PROGRAMMING CONSIDERATIONS .....	4-46
Field Operands .....	4-46
Single Field Operand Instructions .....	4-47
Multiple Field Operand Instructions .....	4-48
Fetching From ISU .....	4-49
Delayed Jumps .....	4-50
Lock On Fetch .....	4-50
Store Operations .....	4-50
Fetch Operations .....	4-51
PROGRAMMING RESTRICTIONS .....	4-51

## CHAPTER IV

# CENTRAL PROCESSOR CHIP (CPC)

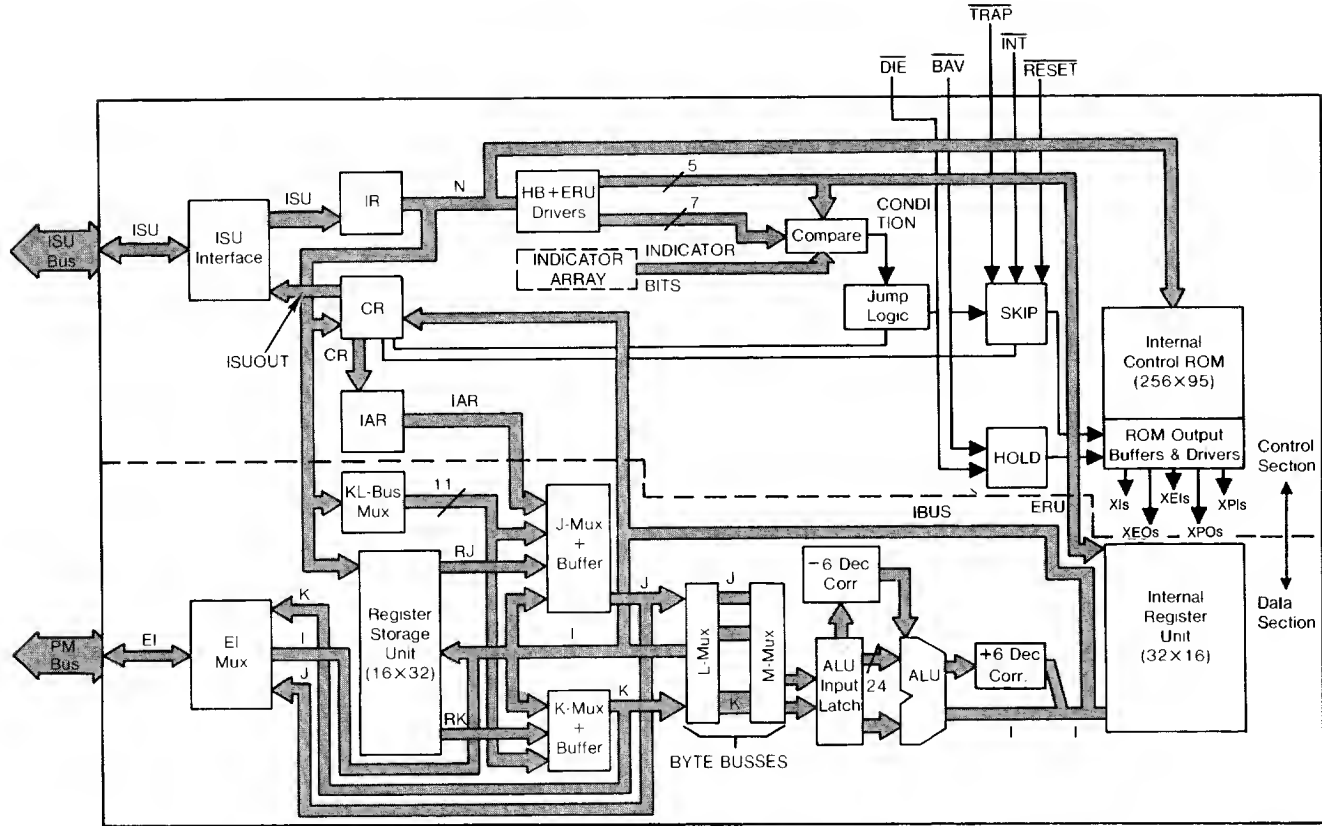
The NCR/32-000 Central Processor Chip (CPC) is an NMOS, 32-bit high performance microprocessor that is externally microprogrammed. This chapter describes the CPC internal architecture and how the CPC interacts with the scratch pad, located in main memory.

### CPC Overview

The CPC is a true 32-bit, externally microprogrammable processor. It contains a set of 179 primitive instructions and variants located in an on-chip ROM (Figure 4-1). These instructions provide the user with a high degree of flexibility to generate microcode programs which are stored in off-chip memory. This memory (organized into 16-bit words) is accessed via the Instruction Storage Unit Bus (ISU Bus). The CPC features a very sophisticated set of operating registers referred to in Figure 4-1 as the Register Storage Unit (RSU). The RSU is organized as sixteen 32-bit registers which, with the ALU, allow the user to perform arithmetic operations at the digit (4-bits), byte (8-bits), half-word (16-bits), word (32-bits) and field (string) levels. The RSU registers can also be organized into even/odd address/data pairs for efficient manipulation of field data. The highly regular structure of the RSU simplifies the effort in writing microcode, while the versatility of the RSU makes emulation of complex virtual machines and execution of complex arithmetic operations easy and efficient. To facilitate op-code cracking of virtual machine instructions, a set of internal registers called "Set-Up Registers" is provided.

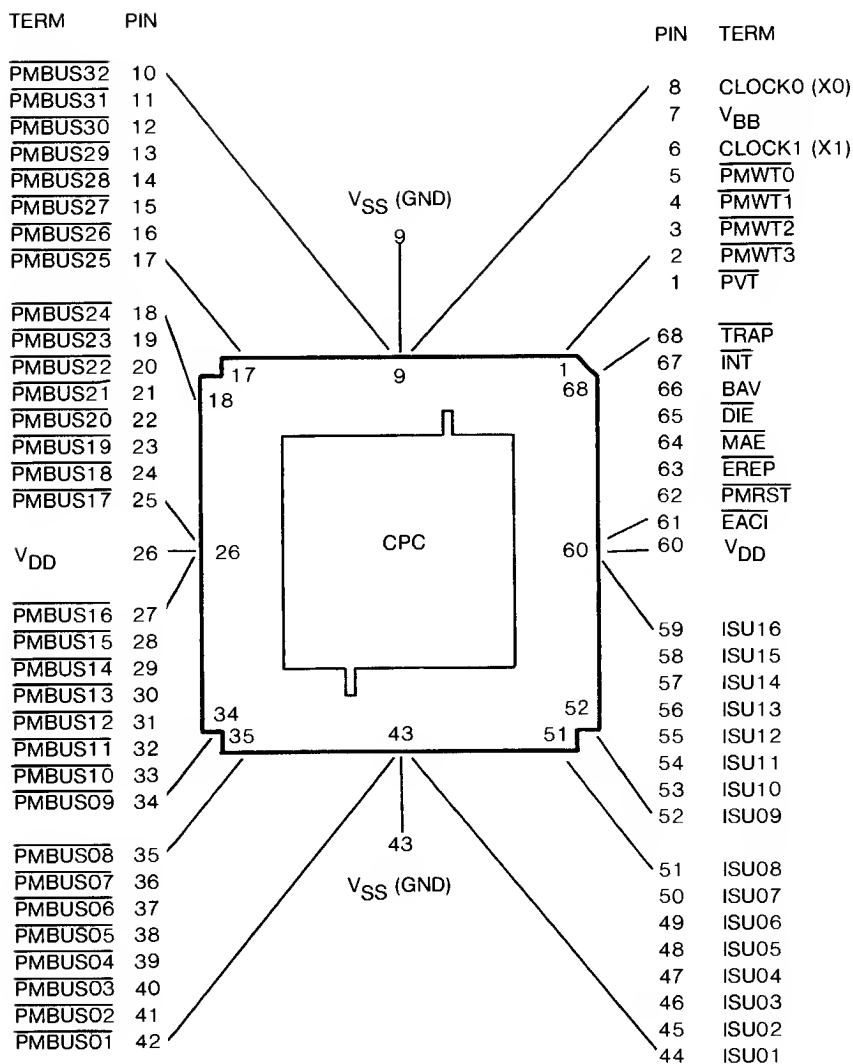
### SIGNAL DESCRIPTION

Table 4-1 gives a brief description of the CPC input and output signals. The input and output signals can be organized into groups as shown in Figure 4-3. CPC pinout is shown in Figure 4-2, and a CPC signal summary is presented in Table 4-2.



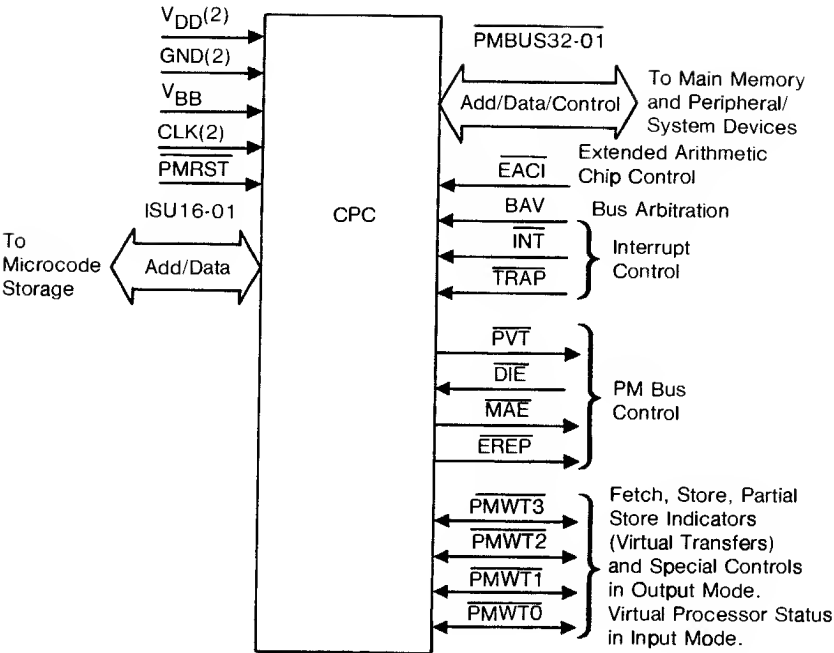
GIM3016AB

Figure 4-1 CPC Functional Block Diagram



GIM4009A

Figure 4-2 CPC Pin Diagram Bottom View (Opposite From Lid)



GIM4010A

Figure 4-3 Input and Output Signals

Pin	Signal	Description
1	$\overline{PVT}$	<p>Processor Virtual Transfer—This active low signal is generated by the processor during the X0 clock phase of all virtual memory operations.</p> <p>This signal is also generated during the X1 clock phase by the processor for all processor-generated memory operations; that is, for all processor generated PM bus operations with the exception of ERU transfers.</p>
2-5	$\overline{PMWT3}$ - $\overline{PMWT0}$	<p>Processor to Memory Write Tags 0-3—These active low signals are generated during the X0 time that <math>\overline{PVT}</math> is generated by the processor. They indicate which bytes are to be written into memory during a virtual transfer. <math>\overline{PMWT3}</math> corresponds to memory byte 3, etc. All Write Tags inactive implies a Fetch operation. During the X1 clock phase, these signals have other definitions:</p> <p><math>\overline{BCT}</math> (Between Commands Testing)—This processor output is asserted on the <math>\overline{PMWT3}</math> pin during every X1 clock phase. It is driven by the BCT bit of Control Array</p>

GIMTE4011B-1

Table 4-1 Pin Description



Pin	Signal	Description
		<p>#1 in the processor. If the BCT bit is set to a 1, then <u>PMWT3</u> will be low each X1.</p> <p><u>VPBSY</u> (Virtual Processor Busy)—The input <u>VPBSY</u> is monitored on the <u>PMWT2</u> pin during the X1 clock phase during the interpret stage of instructions JRPX and JRMX.</p> <p><u>BKPTÉ</u> (Breakpoint Enable)—During the X1 clock phase, this signal is asserted on <u>PMWT1</u>. It is intended to be used to represent the data bit (breakpoint set/reset) to be loaded into an optional off-chip breakpoint RAM if the signal <u>BKPTWE</u> enables modification of the addressed breakpoint. <u>BKPTÉ</u> can be activated by the DJOR and RTI instructions.</p> <p><u>BKPTWE</u> (Breakpoint Write Enable)—This signal is asserted during the X1 clock phase on the <u>PMWT0</u> pin. It is intended to be used as the write enable control for an optional off-chip breakpoint RAM. When this signal is active, it permits the breakpoint RAM to be modified as directed by the <u>BKPTÉ</u> signal. This pin can be activated by the DJOR and RTI instructions.</p>
6	X1	CLOCK1 (X1) phase input.
8	X0	<p>CLOCK0 (X0) phase input.</p> <p>The processor uses two externally-supplied, 2-phase, non-overlapping clocks to control all internal operations. These free-running clocks are X0, the first phase clock, and X1, the second phase clock.</p>
7	VBB	Negative Voltage Supply.
9, 43	VSS	Ground.
10-25 27-42	<u>PMBUS32-</u> <u>PMBUS01</u>	<p>Processor Memory Bus—This active low open drain time-shared bus is the connecting data/address path between the processor and the other NCR/32 Family chips and main memory.</p> <p>ERU addresses are transferred from the processor via the PM bus at X0 times. Memory addresses are transferred from the processor at X0 times. ERU and memory data is transferred to/from the processor at X1 times.</p>
44-59	ISU01- ISU16	<p>Instruction Storage Unit Bus—This is an active high, tri-state bus which links the processor to its ISU memory. During the X0 phase of each clock cycle, the processor outputs an address onto this bus. At the end of the X1 clock phase of the same clock cycle, the processor latches the contents of this bus into its Instruction Register. This is the micro-code instruction or literal stored at the addressed ISU location.</p>

Table 4-1 Pin Description (Continued)

GIMTE4011B-2

Pin	Signal	Description
26, 60	VDD	Positive Voltage Supply.
61	$\overline{\text{EACI}}$	Extended Arithmetic Chip Information—This is a status input to the CPC from the EAC. After the CPC has initiated an EAC operation, the processor can test the EAC Busy flag in the Indicator Array, cleared by an asserted $\overline{\text{EACI}}$ to determine when the EAC has completed the operation.
62	$\overline{\text{PMRST}}$	PM Bus Reset—This asynchronous active low reset signal is generated by the system external reset logic. $\overline{\text{PMRST}}$ is used within the CPC to initialize the processor control logic to an appropriate state.
63	$\overline{\text{EREP}}$	External Register Enable/Permit—This active low processor output is generated by the processor to initiate an ERU transfer. It is generated during the X0 clock phase.
64	$\overline{\text{MAE}}$	Memory Address Enable—This active low signal is generated by the processor during X0 time during real memory address transfers initiated by the processor over the PM bus.
65	$\overline{\text{DIE}}$	Data Input Enable—This active low signal is monitored by the processor at the end of all X1 times following the initiation of a memory fetch sequence to synchronize the processor pipeline to memory data on the PM bus. If $\overline{\text{DIE}}$ is asserted during X1 and meets the required setup and hold times, the processor unlocks the pipeline and latches the data that is asserted on the PM Bus the following X1 clock.
66	BAV	Bus Available—This is an input to the processor; its insertion indicates that the PM bus will be free for the processor to use during the next clock cycle. For processor instructions which do not require the PM bus, this signal is ignored. For instructions which require the use or availability of the PM bus, the processor will halt until BAV is asserted to indicate that it may use the bus in the next cycle.
67	$\overline{\text{INT}}$	Interrupt—The processor monitors the $\overline{\text{INT}}$ signal at the end of all X0 times to determine whether an interrupt is being sourced to the processor.
68	$\overline{\text{TRAP}}$	Trap—The processor monitors the $\overline{\text{TRAP}}$ signal at the end of all X0 times to determine whether a trap is being sourced to the processor.

NOTE: Signals with bars are active-low-true signals.

GIMTE4011B-3

Table 4-1 Pin Description (Continued)

Signal Name	Pin #	Symbol	Input/ Output	Active State	Drive
Processor Virtual Transfer	1	$\overline{\text{PVT}}$	Output	Low	Open Drain
Processor-Memory Write Tag	2,3,4,5	$\overline{\text{PMWT3-0}}$	Input/ Output	Low	Open Drain
Clock 1	6	X1	Input	High	Input
Power Input	7	VBB	Input	—	—
Clock 0	8	X0	Input	High	Input
Ground	9,43	VSS	Input	—	—
Processor-Memory Bus	10-25, 27-42	$\overline{\text{PMBUS32-01}}$	Input/ Output	Low	Open Drain
Instruction Storage Unit Bus	44-59	ISU01-16	Input/ Output	High	3-State
Power Input	26,60	VDD	Input	—	—
Extended Arithmetic Chip Information	61	$\overline{\text{EACI}}$	Input	Low	Input
Processor-Memory Reset	62	$\overline{\text{PMRST}}$	Input	Low	Input
External Register Enable/Permit	63	$\overline{\text{EREP}}$	Output	Low	Open Drain
Memory Address Enable	64	$\overline{\text{MAE}}$	Output	Low	Open Drain
Data Input Enable	65	$\overline{\text{DIE}}$	Input	Low	Input
Bus Available	66	BAV	Input	High	Input
Interrupt	67	$\overline{\text{INT}}$	Input	Low	Input
Trap	68	$\overline{\text{TRAP}}$	Input	Low	Input

GIMTE4012A

Table 4-2 Signal Summary

## DATA ORGANIZATION IN MEMORY

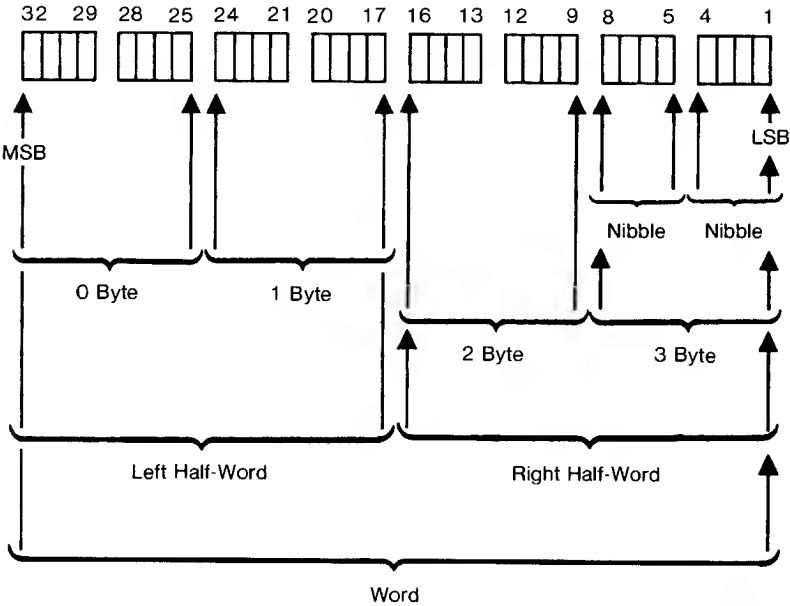
The NCR/32-000 CPC accesses two separate memory arrays. One is the Instruction Storage Unit (ISU), and the other is Main Memory.

### ISU MEMORY

The NCR/32-000 communicates with the ISU memory over the 16-bit multiplexed ISU bus. The ISU memory contains the external microinstruction programs. The microinstruction organization is presented in the Instruction Set chapter of this manual.

MAIN MEMORY

The NCR/32-000 communicates with main memory over the 32-bit multiplexed PM bus. During read operations (Fetch-Receive instruction sequence), an entire 32-bit word is read from memory and placed into a designated RSU location. During store operations, an entire word is written into a memory location. Figure 4-4 shows word organization in memory.



GIM2130A

Figure 4-4 Word Organization In Memory

Main memory is divided into two sections, scratch pad and program/data storage, with the scratch pad located at the upper most 128 words of the address space (Figure 4-5). See the Scratch Pad Access section of this chapter for a more complete description.

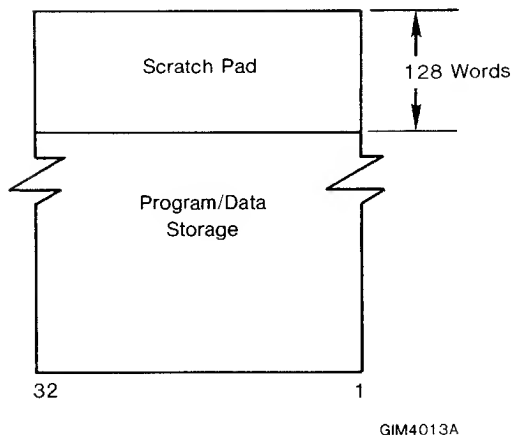


Figure 4-5 Main Memory Organization

### THREE-STAGE PIPELINE

The NCR/32-000 executes microinstructions within a three-stage pipeline consisting of a fetch stage, an interpret stage, and an execute stage.

In the fetch stage, the microinstruction in the ISU memory is addressed from the Control Register (CR) and then read out of memory and loaded into the Instruction Register (IR). Both the CR and IR are internal CPC registers not directly accessible by the user.

In the interpret stage, the microinstruction currently in the Instruction Register is decoded to set up the controls used during the instruction execution and to read out the operands from the Register Storage Unit (RSU) to be used during execution.

In the execute stage, the operands are processed as appropriate to the instruction type. Information is routed internally throughout the CPC and externally over the PM bus. Results are written back into the RSU as required.

### ARITHMETIC LOGIC UNIT (ALU)

The ALU inside the CPC performs binary functions on words and bytes, decimal arithmetic functions on bytes, logical functions on words, bytes, and nibbles, and shift functions on words.

ALU binary arithmetic functions include Add, Subtract, and Compare. Decimal functions include Add and Subtract. Logical functions include AND, OR and Exclusive OR, and Invert. Shift functions include single-bit Shift Right and Shift Left.

## ARITHMETIC OPERATIONS

Decimal operations may be in either packed or unpacked BCD representation. Data bytes in the packed format contain two BCD digits. Data bytes in the unpacked format contain a BCD digit and an ASCII zone character.

Binary operands may be in either a signed or an unsigned format. In the signed format, the most-significant bit of the operand determines the sign of the operand. If the most-significant bit is a zero, the sign is positive. If the most-significant bit is a one, the sign is negative. Operand values for unsigned binary numbers range from 0 to  $(2^{**}N)-1$ , where N equals the total number of bits in the operand. Operand values for signed binary numbers range from  $-2^{**}(N-1)$  to  $-1$  for negative numbers and from  $+0$  to  $+ [2^{**}(N-1)]-1$  for positive numbers, where N again equals the total number of bits in the operand.

During binary subtraction, the subtrahend is two's complemented and then added to the minuend. During binary compare, one operand is two's complemented and then added to the other. A compare operation does not write a result back into the RSU.

Most ALU operations affect the machine indicators (flags). The validity of the indicators is often dependent upon the data representation of a particular instruction. When not applicable, an indicator should be ignored. For example, the Overflow Indicator is set during several binary operations but is valid only for signed binary operations.

The ALU represents negative numbers in two's complement form.

## SPECIAL DECIMAL ALU LOGIC

The digit operands used in decimal field arithmetic operations, field transfers, and digit transfers are checked prior to entering the ALU. If the operand is not in the packed BCD format, then the PBCD indicator is set. If the zone digit is not a valid character (not 0011), then the UBCD indicator is set. These checks are performed a byte at a time and are reset by firmware.

During unpacked decimal arithmetic operations, the ASCII zone character "3" is inserted in the upper digit of the resultant byte. During unpacking operations, the digit operand is tested prior to the transfer. If the digit is nine or less, the ASCII zone character "3" is inserted into the upper digit of the resultant byte. If the digit is greater than nine, the ASCII zone character "2" is inserted into the upper digit of the resultant byte.

## REGISTER DESCRIPTION

The NCR/32-000 features two sets of internal registers available to the user: Register Storage Unit and the Internal Register Unit. The Register Storage Unit (RSU) consists of sixteen 32-bit registers that are used for data and address storage and manipulation. The Internal Register Unit (IRU) consists of twenty-two special registers used for status information, jump addresses, field byte count information (Tally Register), and special set-up for virtual machine emulation. The CPC can also access 96 peripheral and user-defined external registers called ERUs.

### REGISTER STORAGE UNIT (RSU)

The RSU consists of sixteen 32-bit registers. All non-literal instruction operands are read from the RSU during the interpret phase of instruction execution. Results of an operation are written back into the RSU during the Execution phase.

During most operations, the RSU assignments are not instruction dependent. During Field operations and Store operations, however, adjacent even and odd RSUs are paired together as Memory Assist Registers (MARS). RSU0 becomes MARS0 Address Register and RSU1 becomes MARS0 Data Register, RSU2 becomes MARS1 Address Register and RSU3 becomes MARS1 Data Register, etc. A Field or Store instruction will specify a MARS Data Register which implies the use of the associated MARS Address Register. For Virtual Fetch operations, the Data RSU must always be an odd-numbered RSU.

The two MARS7 RSUs are generally reserved as the Virtual Control Register (used as a program counter for virtual instructions) and the Virtual Instruction Register (used to hold the virtual instruction addressed by the Virtual Control Register portion of MARS7), but otherwise function like the other RSUs.

All RSU entries are accessible on a word basis through the J and K operand address fields of the micro-instruction. The first four RSUs (RSU0-RSU3) are also addressable to the byte level.

RSU9, RSU11, RSU13, and RSU15 are indirectly byte addressable; that is, bytes are not directly addressable via micro-instruction but rather are addressed via the Byte Pointers. The Byte Pointers are 2-bit binary counters which correspond to the two least-significant bits of RSU8 (MARS4 Address Register), RSU10 (MARS5 Address Register), RSU12 (MARS6 Address Register), and RSU14 (MARS7 Address Register). The Byte Pointers are loaded when the MARS Address Register is loaded, and subsequently incremented or decremented during Field instructions (or Setup

instructions for MARS7). The RSU registers and their characteristics are shown in Table 4-3.

RSU #	Transfers				MARS Function	Memory Access			Protection Checks				Field Usage
	W	HW	B2	D		WF	WS	BS	R	W	L	E	
0	X	X	X	X	MARS0 Addr	X	X	X	X	X	X		
1	X	X	X	X	MARS0 Data	X	X	X	X	X	X		
2	X	X	X	X	MARS1 Addr	X	X	X	X	X	X		
3	X	X	X	X	MARS1 Data	X	X	X	X	X	X		
4	X	X			MARS2 Addr	X	X	X	X	X	X		
5	X	X			MARS2 Data	X	X	X	X	X	X		
6	X	X			MARS3 Addr	X	X	X	X	X	X		
7	X	X			MARS3 Data	X	X	X	X	X	X		
8	X	X			MARS4 Addr	X	X	X	X	X	X	Operand Fetch	
9	X	X	X		MARS4 Data	X	X	X	X	X	X		
10	X	X			MARS5 Addr	X	X	X	X	X	X	Operand Fetch	
11	X	X	X		MARS5 Data	X	X	X	X	X	X		
12	X	X			MARS6 Addr	X	X	X	X	X	X	Operand Store	
13	X	X	X		MARS6 Data	X	X	X	X	X	X		
14	X	X			MARS7 Addr	X	X	X			X	V Cont	Reg V Inst Reg
15	X	X	X		MARS7 Data	X	X	X			X		

B = Byte

HW = Halfword

WF = Word Fetch

D = Digit

L = Linkage

WS = Word Store

E = Execute

R = Read

BS = Byte Store

1. Byte transfers on RSU 9, 11, 13, and 15 are during field instructions only.
2. 32-bit words can be fetched from main memory and placed into any of the RSU locations as specified by the J-field and K-field of the micro-instruction. Full or partial word stores can be used to place data into main memory from any RSU location.
3. On virtual transfers, the two least-significant bits of the PM bus (PMBUS02 and 01) contain the appropriate protection code. The Fetch for Execution protection is invoked on accesses with RSU 14 and 15.

PMBUS02 PMBUS01

### Function

0	0	Fetch
---	---	-------

0	1	Potion Store
---	---	--------------

1	0	Fetch for Linkage
---	---	-------------------

1	0	Fetch for Linkage
1	1	Fetch for Execution

The PMBUS02 and PMBUS01 values represent logic levels (i.e., logical “0” represents a high voltage level on the signal line and a logical “1” represents a low voltage level on the signal line).

GIMTE4014A

Table 4-3 RSU Characteristics



## EXTERNAL REGISTERS (ERUs)

The CPC can address up to 128 register assignments (Table 4-4) via special transfer instructions. The first 32 (address locations 00-1F hex) are contained inside the CPC and are referred to as Internal Registers (IRUs). The RSU previously described and the IRU represent all of the user accessible registers inside the CPC. The IRU is described in detail in the following section of this chapter.

The remaining 96 register assignments represent register locations external to the CPC (e.g. Scratch Pad, I/O Ports, or Registers inside the Address Translation Chip - ATC) and are referred to as External Registers (ERUs).

	7-bit Addr. (DEC) (HEX)		Function	Transfer Instructions
IRUs	00-07	00-07	Jump Registers	TII & TOI
	08	08	Restore FIFO	
	09	09 1	State Register	
	10	0A	Setup Register #3	
	11	0B	Setup Register #4	
	12	0C	Tally Register	
	13-15	0D-0F	Not Used	
	16	10	Indicator Array	
	17	11	Virtual Indicators	
	18	12	Setup Register #5	
	19	13	Setup Register #1	
	20	14	Setup Register #2	
	21-23	15-17 2	Not Used	
	24	18	Operand Pointer #1 and #2	
	25	19	Not Used	
	26	1A	Stack Pointer	
	27	1B	Control Array #1	
	28	1C	MARS6 Write Tags	
	29-31	1D-1F	Not Used	
ERUs	32	20	Operand Data 1	TIE & TOE
	33	21	Operand Data 1 Inc	
	34	22	Operand Data 1 Dec	
	35	23	Operand Data 2	
	36	24	Operand Data 2 Inc	
	37	25	Operand Data 2 Dec	
	38	26	Stack Data	
	39	27	Not Used	
	40	28	Control Array #2	
	41	29	Interrupt/Trap Array/	
	42	2A	Interrupt Mask	
	43	2B	Interval Monitor	
	44	2C	Time of Day	
	45	2D	Address Monitor	
	46	2E	BIN Register	
	47-55	2F-37	Associative Memory Functions	
	50-63	38-3F	Reserved for Future Use	
	64-127	40-7F	Reserved for Future Use	TIP & TOP

Table 4-4 Internal Register (IRU) and External Register (ERU)  
Address Assignments

1. These are 16-bit Right Justified IRU registers. A TII (Transfer In Internal) from one of these registers does not affect the left-half of the destination RSU.
2. These are 16-bit Left Justified registers. A TII from one of these registers does not affect the right-half of the destination RSU.
3. Registers 00-31 (00-1F Hex) are registers internal to the CPC, hence, are referred to as Internal Register Units (IRU). They are accessed via the TII and TOI instructions.
4. Registers 32-55 (20-37 Hex) are registers external to the CPC, hence, are referred to as External Register Units (ERU). These registers all have assigned functions. They are accessed via the TIE and TOE instructions.
5. Registers 56-63 (38-3F Hex) are ERUs reserved for future NCR 32-000 family devices.
6. Registers 64-127 (40-7F) are also ERUs but do not belong to the basic Processor Subsystem. These ERUs do not have fixed functional assignments but rather may be grouped together to form a collection of addressable PM Bus registers or "ports," on the PM bus. These registers are accessed via the TIP and TOP instructions.
7. Instructions which transfer information to or from an external register over the Processor-Memory Bus supply the seven-bit external register number via the seven least significant bits of the Processor-Memory Bus. The direction of transfer is specified on PM Bus 08 (active = external register receives data, inactive = external register sources data).

GIMTE4015

Table 4-4 Internal Register (IRU) and External Register (ERU)  
Address Assignments (Continued)

## INTERNAL REGISTER UNIT (IRU)

There are thirty-two internal register locations inside the NCR/32-000 which are referenced by the Transfer Out Internal (TOI) and Transfer In Internal (TII) instructions (Figure 4-6). These registers are described in the following paragraphs.

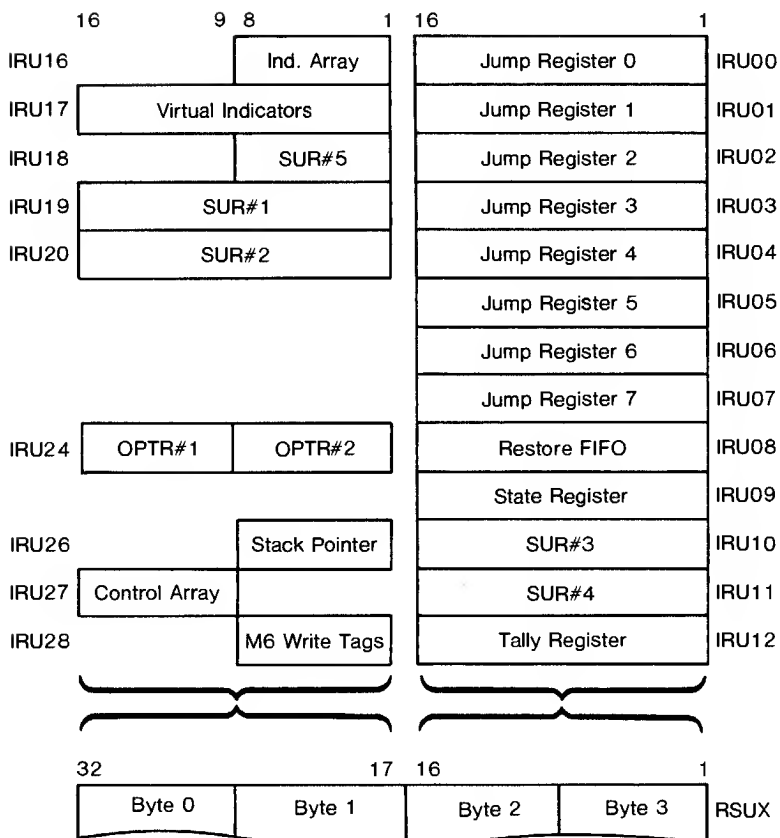
### Jump Registers (IRU0-IRU7)

There are eight addressable 16-bit registers that can be used to hold jump addresses. These registers are addressable as internal registers (IRU0-7) and through the J-field of certain jump instructions.

During the Jump on Field Array instruction, Jump Register 7 is the source register containing the base jump vector. Jump Register 6 is the destination register receiving the link address.

Firmware may choose by convention to dedicate some of the Jump Registers that contain frequently used addresses (e.g., the address of the BCT flow).

A Transfer Out to a Jump Register loads the least-significant sixteen bits of an RSU into the register. A Transfer In from a Jump Register loads the contents of the Jump Register into the least-significant sixteen bits of an RSU without altering the upper sixteen bits.



IRU00-12 May be Transferred Into or Out  
From the Right Halfword of an RSU  
Without Disturbing the Left Halfword

IRU16-20, 24, 26-28 May be Transferred  
Into or Out From the Left Halfword  
of an RSU Without Disturbing the  
Right Halfword

Unused IRUs Are Not Physically  
Implemented in the CPC

GIM3021-2A

Figure 4-6 CPC Internal Register Unit

“Transfer Out” and “Transfer In” are partial CPC instruction names which indicate write and read operations to/from registers and memory. The complete names further specify whether the operations are to “internal” or “external” register/memory locations. See Chapter VI of this manual for further information on these OP Codes.

### **Restore Field (IRU8)**

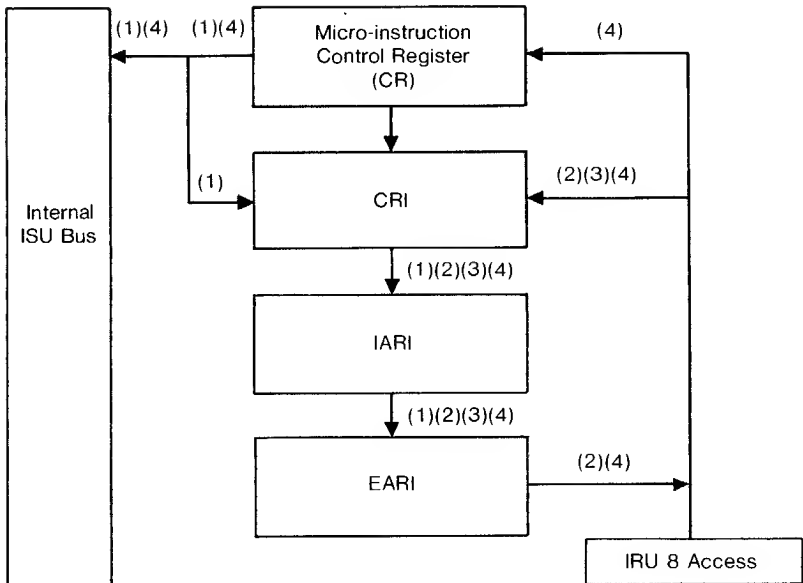
The Restore FIFO is a 3-deep, 16-bit wide, first-in first-out shift register connected to the micro-instruction Control Register. The FIFO saves the addresses of all three instructions that are in the pipeline during normal program execution (Execution Address in EARI, Interpret Address in IARI, and Control Register value in CRI). When an interrupt or trap is taken, the Restore FIFO clocking is stopped until a Restore from Interrupts/Traps instruction (RTI) is executed with the Restart FIFO control bit set. The FIFO may also be stopped by executing an RTI instruction with the Restart FIFO control bit clear.

When the Restore sequence (three RTI instructions) is executed at the end of an interrupt or trap service routine, the three addresses saved in the FIFO are sequentially injected into the micro-instruction Control Register to restart the normal program flow.

The Restore FIFO may be read using the Transfer In instruction and loaded using the Transfer Out instruction. A Transfer In always reads the bottom-most entry (EARI) into the least-significant sixteen bits of an RSU without altering the upper sixteen bits. At the completion of the Transfer In, the FIFO is shifted one position; that is, CRI moves to IARI, IARI moves to EARI, and EARI moves to CRI. If the Transfer In is executed during normal program flows, the EARI moving to CRI overrides the CR transfer to CRI that normally occurs during execution. Care should be taken when accessing the FIFO during normal program execution.

A Transfer Out always loads the least-significant sixteen bits of an RSU into the top-most entry (CRI) of the FIFO. At the completion of the Transfer Out CRI has been loaded from an internal bus, IARI has been loaded from CRI, and EARI has been loaded from IARI. If the Transfer Out is executed during normal program flows, the loading of the CRI from the internal bus overrides the loading of CRI from CR.

Figure 4-7 shows the Restore FIFO and its interface to the micro-instruction Control Register and the internal bus.



1. Normal flow
2. Transfer in
3. Transfer out
4. RTI

GIM4016A

Figure 4-7 Restore FIFO

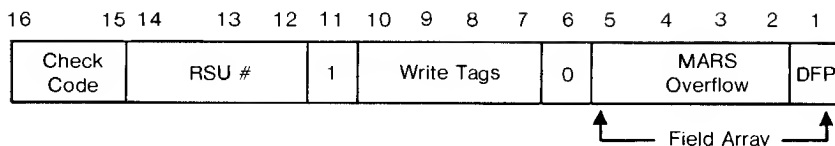
### State Register (IRU9)

The State Register is a 16-bit status register used to save key processor control information during the execution of Virtual Memory instructions. That information is then used during subsequent MRR (Memory Reference Retry) instruction execution. The State Register is also loaded from the least-significant sixteen bits of an RSU with a Transfer Out instruction and, conversely, is read into the least-significant sixteen bits of an RSU (with the most-significant sixteen bits not affected) with a Transfer In.

### CAUTION

Care should be taken when writing to this register to prevent altering needed processor status. When resetting certain bits, it is good programming practice to leave the remaining bits of the register unchanged.

The State Register is formatted as follows:



- Bit 1: Direction of Field Processing (DFP). Left-to-Right (0). Right-to-Left (1).  
 Bit 2: MARS4 Overflow Flag  
 Bit 3: MARS5 Overflow Flag  
 Bit 4: MARS6 Overflow Flag  
 Bit 5: MARS7 Overflow Flag  
 Bit 6: Not Used = 0  
 Bit 7: Memory Byte 3 Write Tag  
 Bit 8: Memory Byte 2 Write Tag  
 Bit 9: Memory Byte 1 Write Tag  
 Bit 10: Memory Byte 0 Write Tag  
 Bits 14-11: RSU Address; Bit 11 = 1  
 Bits 16, 15: Protection Check Code

GIM4017

Figure 4-8 State Register Format

### Bits 1-5 — Field Array Bits

The least-significant five bits of the State Register are the Field Array bits which are concatenated with Jump Register 7 to form the jump vector during Jump on Field Array execution. The Field Array consists of the Overflow Flags for MARS4-7, and the Direction of Processing Indicator.

The Overflow Flags are automatically set for the respective MARS units when the corresponding Byte Pointers in the RSU cross the word boundary during a Field instruction. When using word-aligned fields, as many as three Overflow Flags can set simultaneously. The Overflow Flags should be cleared by firmware before exiting the service routine. The Direction of Processing Indicator is automatically modified during execution of a Field instruction to reflect left-to-right or right-to-left memory data processing. If no specific direction is indicated in the instruction, then left-to-right processing is used (0 = left-to-right; 1 = right-to-left). This bit can be modified by firmware (TOI instruction) or execution of another Field instruction.

### Bits 7-10 — Byte Write Tag Bits

Bits 7-10 of the State Register are a copy of the Memory Byte Write Tags which control byte writing during Memory Store operations.

The Byte Write Tags in the State Register are a copy of the corresponding bits in the internal Write Tag Register. The Write Tag Register is an internal holding register not accessible by the user.

The contents of this register are used to drive the appropriate PM bus lines during certain store operations (refer to Figure 4-12). The Write Tag Register is copied into bits 7-10 of the State Register during all Virtual Store instructions for use during a subsequent Memory Reference Retry when an error condition, such as a DAT translation error, is detected during a Virtual Store. The Byte Write Tags do not change until another Virtual Store is executed.

### **Bits 11-14 — RSU Address Bits**

Bits 11-14 of the State Register are the RSU Address bits which are used to specify the source or destination RSU during Memory Reference Retry operations.

Virtual Store instructions and Receive Fetched Data instructions (which have bit 1 of the K-field set) enable the address specifying the Data RSU to be saved in bits 11-14 of the State Register. Bit 11 is forced to a one, which forces the use of an odd-numbered register. Subsequently, if a DAT interrupt occurs during a Virtual Store or Virtual Fetch operation, the saved address will be used during the memory retry portion of the interrupt service routine. The Memory Reference Retry (MRR) instruction uses the RSU specified by the State Register as the Source Data Register when performing a Store operation. The Receive Fetched Data instruction (which has bit 2 of the K-field set) used in conjunction with an MRR instruction uses the RSU specified by the State Register as the Destination Data Register when performing a Fetch operation.

### **Bits 15-16 — Protection Check Code Bits**

Bits 15 and 16 of the State Register are automatically written during all virtual memory transfers with the Protection Check Code bits which are asserted on PMBUS01,02. Bit 15 corresponds to PMBUS01, and bit 16 to PMBUS02.

During the execution of an MRR instruction, bits 15 and 16 are then used as the Protection Check Code for the retry memory operation.

A decode of the Protection Check Code is used by the hardware to distinguish between Store (01) operations and Fetch (not 01) operations. The Protection Check Codes are defined in Table 4-5.

PMBUS02 Logical Level	PMBUS01 Logical Level	Function
0	0	Read (Fetch)
0	1	Write (Store)
1	0	Linkage (Fetch)
1	1	Execute (Fetch via MARS7)

NOTE: Logical 1 level is a low voltage level (active) on the PM Bus and a logical 0 is a high voltage level (inactive) on the PM Bus

GIMTE4018A

Table 4-5 Protection Check Code

### Indicator Array (IRU16)

The Indicator Array is an 8-bit register used to identify the state of the processor, particularly the ALU, following an instruction execution. The Indicator Array can be loaded from byte 1 of an RSU with a Transfer Out instruction and, conversely, can be read into byte 1 of an RSU (with byte 0 zero-filled and bytes 2 and 3 not affected) with a Transfer In instruction.

Not all instructions affect the Indicator Array. For some of the instructions that affect the Array, the resulting state of the indicators is not defined. Also, the indicators may be defined differently for one instruction than for another. Table 4-6 shows the defined bits for various categories of instructions.

Conditional Jump and Conditional Skip instructions test the Indicator Array bits. The specific bits to be tested are selected by the setting of the Bit Pair Selector and the Bit Pair Mask in the microinstruction. Refer to "Condition Selector" in the Microinstruction Set (Chapter VI) for details.

The following definitions apply to the indicators in Table 4-6.

#### 1. <ZERO, =ZERO, >ZERO

These indicators are set/reset during most arithmetic and boolean instructions to reflect whether the result of an operation was less than, equal to, or greater than zero. The indicators do not change until a compare, arithmetic, or boolean instruction that has been specified to modify these indicators is executed.

#### 2. LESS, EQUAL, GREATER

These indicators are set/reset during all compare instructions to reflect whether one operand was less than, equal to, or greater than another operand. The indicators do not change until a compare, arithmetic, or boolean instruction is executed.

#### 3. CARRY

This indicator is set/reset during all arithmetic instructions and during some shift instructions. It reflects that a carry of the most-significant bit position of the result has occurred during the operation (exception: Shift Right instruction where the carry occurs from the least-significant bit position). For binary



subtraction, a carry represents no borrow, and no carry represents a borrow. This indicator does not change until another arithmetic or shift instruction is executed. Firmware must initialize this indicator prior to certain instruction executions.

#### 4. OVERFLOW

This indicator is set/reset during most binary arithmetic instructions. It reflects that, as a result of an operation, a carry was forced into the sign position (most-significant bit) when the sign bit should have been a zero, or a carry-out occurred from the sign position when the sign bit should have been a one. This indicator does not change until an arithmetic, a field compare, or a digit transfer instruction is executed.

#### 5. PBCD

This indicator is set during field decimal arithmetic, field compare, field transfer, and digit transfer instructions. It reflects that a non-numeric digit was detected during the operation. This indicator does not change once set until cleared by firm ware.

#### 6. UBCD

This indicator is set during field decimal arithmetic, field compare, field transfer, and digit transfer instructions. It reflects that an illegal ASCII zone character (not 0011) was detected. This indicator does not change once set until cleared by firm ware.

#### 7. EAC BUSY

This indicator is set by the Extended Arithmetic Function instruction. It reflects that an operation has been initiated in the EAC and that the EAC is busy. The indicator is reset when the EAC signals (via  $\overline{\text{EACI}}$  pin) the processor that the operation is complete and the results are ready. EAC Busy is not affected by a TOI to IRU16, but can be read via a TII from IRU16.

#### 8. C.C. MATCH

This indicator is set during the SETNA instruction on branch formats when the condition code field of the Virtual instruction matches the Virtual indicators. The indicator is reset upon entering a subsequent SETNA instruction.

During JMPIB or JMPIC instructions, this indicator is set when the condition code field of the IBM instruction matches the Virtual indicators. The indicator is reset or set again during the next JMPIB or JMPIC execution.

#### 9. BCT

This indicator is set/reset by firmware via the TOI to IRU27, SC, and RC instructions. BCT (Between Commands Testing) in the Indicator Array is a copy of bit 9 of Control Array #1. A TOI to IRU16 does not affect this bit. The BCT bit can be read with a TII from IRU16.

Bits	Binary Arithmetic	Decimal Arithmetic	Boolean	Compare	Shift
1	< ZERO	< ZERO	< ZERO	LESS	—
2	= ZERO	= ZERO	= ZERO	EQUAL	—
3	> ZERO	> ZERO	> ZERO	GREATER	—
4	CARRY	CARRY	—	—	CARRY
5	OVERFLOW	PBCD	—	PBCD	—
6	—	UBCD	—	UBCD	—
7	—	—	—	—	—
8	—	—	—	—	—
Bits	Transfer	NG Setup	Extended Arithmetic	SCAL RCAL	SCO RIZ
1	—	—	—	—	—
2	—	—	—	—	—
3	—	—	—	—	—
4	—	—	—	—	CARRY
5	PBCD	—	—	—	PBCD
6	UBCD	C.C.MATCH	—	—	UBCD
7	—	—	EAC BUSY	—	—
8	—	—	—	BCT	—

— Unaffected

GIMTE4019

Table 4-6 Indicator Array Bit Definition

### Virtual Indicators (IRU17)

The Virtual Indicator Register is a 16-bit register which contains various virtual machine level indicators or flags that are maintained by the firmware. Some specific indicators are explicitly defined for each of the virtual machines that is supported. These indicators are the most frequently referenced indicators and are mapped from the Indicator Array (IRU16) via the special Map Indicator instructions.

A Transfer Out to the Virtual Indicators loads all sixteen bits from the most-significant sixteen bits (byte 0, byte 1) of the source RSU. A Map Indicator instruction loads only those bits specified in the instruction. A Transfer In from the Virtual Indicators reads the 16-bit register into the upper half of the destination RSU but does not affect the lower half (bytes 2, 3).

The defined indicators under hardware support are shown in Figure 4-9 for each of the virtual machines.

### Tally Register (IRU12)

The Tally Register is a 16-bit counting register which normally counts the number of bytes to be processed during a virtual command emulation. During the execution of Field instructions or the

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	
										O	R		G	E	L	VRX
										O			G	E	L	NVM
														C 1	C O	IBM

C = Carry

L = Less

C0 = Condition Code 0

O = Overflow

C1 = Condition Code 1

R = Repeat

E = Equal

G = Greater

GIM4020

Figure 4-9 Virtual Machine Indicators

Delayed Jump on Tally instruction, the content of the Tally Register is decremented by one (two if half-word instruction) for each instruction cycle. When the Tally Register is decremented to a zero value, TALLY=0, the Field instruction is exited.

The Tally Register is loaded during virtual command setup by the Load Tally From Setup (LTS) instruction, and can be loaded with an instruction literal by the Load Tally Right and Clear Left (LTRC) instruction. Bit 9 of the Tally Register is loaded with a one if TALLY=0 during the JPMVC instruction.

A Transfer Out to the Tally Register loads sixteen bits from the least-significant sixteen bits of the source RSU. A Transfer In from the Tally Register reads the sixteen bits of the register into the lower half of the destination RSU but does not affect the upper half.

## Operand Pointers (IRU24)

The Operand Pointer Register contains both Operand Pointer #1 (upper byte) and Operand Pointer #2 (lower byte). Both pointers operate the same functionally. They differ only in their applications during virtual setup assist. The Operand Pointers are 8-bit registers used to access information located in the scratch pad portion of main memory. The least-significant seven bits of the registers are a modulo-128 up/down counter which points to one of the 128 scratch pad entries. The eighth bit is an override control which, when set, voids the use of the Operand Pointer and instead enables the Stack Pointer for all scratch pad accesses normally performed using the Operand Pointer. See the Scratch Pad Access section for more details.

Pointer #1 is loaded by a Transfer Out with byte 0 of the source RSU, while Pointer #2 is loaded with byte 1. Conversely, a Transfer In reads the eight bits of the Pointer #2 into byte 1 of the destination RSU, while the eight bits of Pointer #1 are read into byte 0 with bytes 2 and 3 not affected.

During virtual command setup, the Operand Pointers are loaded to reference the virtual registers in the scratch pad that apply to the particular virtual machine being supported. Refer to the "Setup Assist" section for an example description of the modes in which the Operand Pointers can be initialized and subsequently manipulated.

### **Stack Pointer (IRU26)**

The Stack Pointer is a 5-bit, modulo-32 up/down counter which points to one of the 32 operand stack entries in the scratch pad portion of main memory (locations 64-95 decimal).

The Stack Pointer can be loaded by a Transfer Out with the least-significant five bits of byte 1 of the source RSU. A Transfer In from the Stack Pointer reads the 5-bit register into the least-significant five bits of byte 1 of the destination RSU. The remaining bits in byte 1 and all of byte 0 are cleared. Bytes 2 and 3 of the RSU are not altered.

### **Setup Registers**

While the RSU can be used exclusively for the emulation of virtual commands, a special set of registers called Setup Registers have been designed for ease in cracking the opcodes of certain instruction sets. The Setup Register operations have been specifically tailored for the emulation of the IBM 370 instruction set as well as the VRX and NVM (NCR operating systems) instruction sets. The Setup Registers allow the programmer to crack virtual machine instructions and automatically branch to the appropriate micro-code subroutine with a high degree of efficiency. Even though these registers have been designed for specific virtual machine emulation, their function is general in nature, and can be used to crack many other instruction sets.

Before using the RSU registers for emulation of instruction sets other than those mentioned above, the user should make use of the Setup Registers where appropriate. This section describes the functional operation of the Setup Registers (SUR1-5). The "Setup Assist" section of this chapter describes the use of these registers.

**Setup Register #1 (IRU19)** — Setup Register #1 (SUR1) is a 16-bit register used to hold all or part of a virtual instruction for the purpose of decoding the virtual opcode to form a vectored jump to a command setup or command execution routine, and for the purpose of isolating fields within the instruction for subsequent transfers to an RSU.

SUR1 is loaded and interpreted by the special setup instructions as described in the "Setup Assist" section. SUR1 is also loaded by a Transfer Out from the most-significant sixteen bits of a source RSU, and read by a Transfer In to the most-significant sixteen bits of a destination RSU (the least-significant bits of the RSU are undisturbed).

**Setup Register #2 (IRU20)** — Setup Register #2 (SUR2) is a 16-bit register used to hold part of a virtual instruction during NVM or IBM setup for the purpose of loading an Operand Pointer and for isolating the displacement field for a subsequent transfer to an RSU.

SUR2 is loaded and interpreted by the setup instructions as described in the "Setup Assist" section. SUR2 is also loaded by a Transfer Out from the most-significant sixteen bits of a source RSU, and read in by a Transfer In to the most-significant sixteen bits of a destination RSU (the least-significant bits of the RSU are not affected).

**Setup Register #3 (IRU10)** — Setup Register #3 (SUR3) is a 16-bit register that contains three base address values which are concatenated with vectors formed during the setup instructions to perform program jumps to additional command setup or command execution flows.

SUR3 is used by the setup instructions as described in the "Setup Assist" section. SUR3 is loaded by a Transfer Out from the least-significant sixteen bits of a source RSU, and read in by a Transfer In to the least-significant sixteen bits of a destination RSU (the most-significant sixteen bits of the RSU are not affected).

**Setup Register #4 (IRU11)** — Setup Register #4 (SUR4) is a 16-bit register which contains the address of the MARS7 Overflow Fetch routine used during NVM or IBM emulation. In support of NVM, SUR4 is also used as a portion of the base address for program jumps during Descriptor operations.

SUR4 is used by the setup instructions as described in the "Setup Assist" section. SUR4 is loaded by a Transfer Out from the least-significant 16 bits of a source RSU, and read by a Transfer In to the least-significant sixteen bits of a destination RSU (the most-significant sixteen bits are not affected).

**Setup Register #5 (IRU18)**—Setup Register #5 (SUR5) is an 8-bit register which holds the virtual tally that is loaded into the Tally Register with the Load Tally From Setup instruction.

SUR5 is loaded from the virtual instruction during the special setup instructions. SUR5 is also loaded by a Transfer Out from byte 1 of a source RSU, and read by a Transfer In to byte 1 of a destination RSU with byte 0 zero-filled and bytes 2 and 3 not affected.

### Control Array #1 (IRU27)

Control Array #1 is an 8-bit register which contains control bits that are used within the processor.

A Transfer Out to Control Array #1 loads all eight bits from the most-significant eight bits (byte 0) of the source RSU. A Transfer In from Control Array #1 reads the 8-bit register into the upper byte of the destination RSU with zeros loaded into byte 1, but does not affect the lower half-word. Figure 4-10 shows Control Array #1 format.

16	15	14	13	12	11	10	9
ESC	EACA	NUM	AT	TI	NIE	BCT	

GIM4021A

Figure 4-10 Control Array #1 Format

The following are the Control Array #1 bit definitions.

- Bit 9**                      **Between Commands Testing (BCT)**
- This indicator is used by the setup instructions. If BCT is set, the setup instructions branch to the BCT flow (refer to the individual setup commands for branch flow). This indicator is set by firmware (via TOI or SC) whenever there is a need for between commands testing. It must be cleared by firmware when the testing is not desired (via TOI or RC).
- Bit 10**                     **Normal Interrupt Enable (NIE)**
- This indicator is used to enable or disable interrupts. Interrupts are recognized when this bit is set. This indicator is set/cleared by the TOI, SC, and RC instructions, and set by the RTI instruction. An active interrupt condition ( $\overline{\text{INT}}$  pin low) clears this bit.
- Bit 11**                     **Trap Indicator TI**
- This indicator is set when the  $\overline{\text{TRAP}}$  pin is active (low) and forces a jump to the trap address (0000 hexadecimal). This bit, when set, inhibits all pending

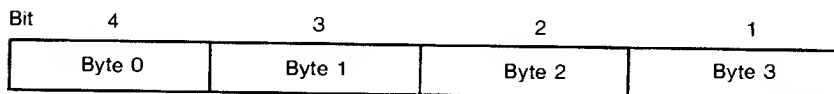
traps and interrupts. The Trap indicator is set/cleared with the TOI, SC, and RC instructions and cleared with the RTI instruction.

- Bit 12**                      **Address Translation (AT)**  
 This bit selects the formatting of memory messages to be in the virtual address format (AT set) or the real address format (AT cleared). AT is set/cleared with the TOI, SC, and RC instructions.
- Bit 13**                      **Non-User Mode (NUM)**  
 This is a test bit not intended for user application and must remain cleared. On power-up, this bit is cleared, and should not be programmed to a 1.
- Bit 14**                      **EAC Activated (EACA)**  
 The EACA bit is intended for use with the Extended Arithmetic Chip. This indicator is set whenever an EAF instruction is executed to indicate that the EAC has been activated. EACA is reset by firmware with a TOI or an RC instruction.
- Bits 15, 16**                **Execution Skip Count (ESC)**  
 At the time an interrupt or trap occurs, the 2-bit skip count is saved in bits 15 and 16. Subsequently, during the execution of an RTI instruction, bits 15 and 16 are loaded into the skip counter. A skip count of two (CA16=1, CA15=0) forces the next two execution cycles to be voided. A skip count of one (CA16=0, CA15=1) forces the next single execution cycle to be voided. The skip count saved in bits 15 and 16 is not altered after being transferred to the master skip control.

### **MARS6 Write Tags (IRU28)**

The MARS6 Write Tags are contained in a 4-bit register (MSB = byte 0 Write Tag, LSB = byte 3 Write Tag), shown in Figure 4-11, that is program accessible for purposes of saving/restoring the machine state. Each data byte loaded into the MARS6 Data Register during field operations sets a corresponding MARS Write Tag. Then during a subsequent Store operation, the MARS6 Write Tags are used if so specified by the J-field of the instruction. The MARS6 Write Tags are cleared when a MARS6 Store instruction executes.

A Transfer Out to the MARS6 Write Tags loads the register from the least-significant four bits of byte 1 of the source RSU. A Transfer In from the MARS6 Write Tags reads the register into the four least-significant bits of byte 1, with the other bits in byte 1 and all of byte 0 zero-filled. Bytes 2 and 3 of the destination RSU are not affected.



GIM4022

Figure 4-11 MARS6 Write Tag Register Format

## WRITE TAG OPERATION

A 4-bit internal register, the Write Tag Register, contains the flags that are normally used to form the byte write enables (PMBUS28-25) during all real memory store messages, or the corresponding processor-memory write tags (PMWT0-3) during all virtual memory store messages. This register is not accessible as an internal register, although a copy of it is maintained during normal program emulation in the State Register during Virtual Store operations.

During Store instructions, this internal register is loaded from the J-field and is used as the source of the write tags if at least one J-field bit is on. The four J-field bits specify the memory byte write tags that will be set when the Store operation is performed. If none of the J-field bits are set, then the MARS6 Write Tag Register (IRU28) is used.

During the Store Literal instruction, this internal register is not affected. The memory byte write tags are all set for the Store Literal instruction.

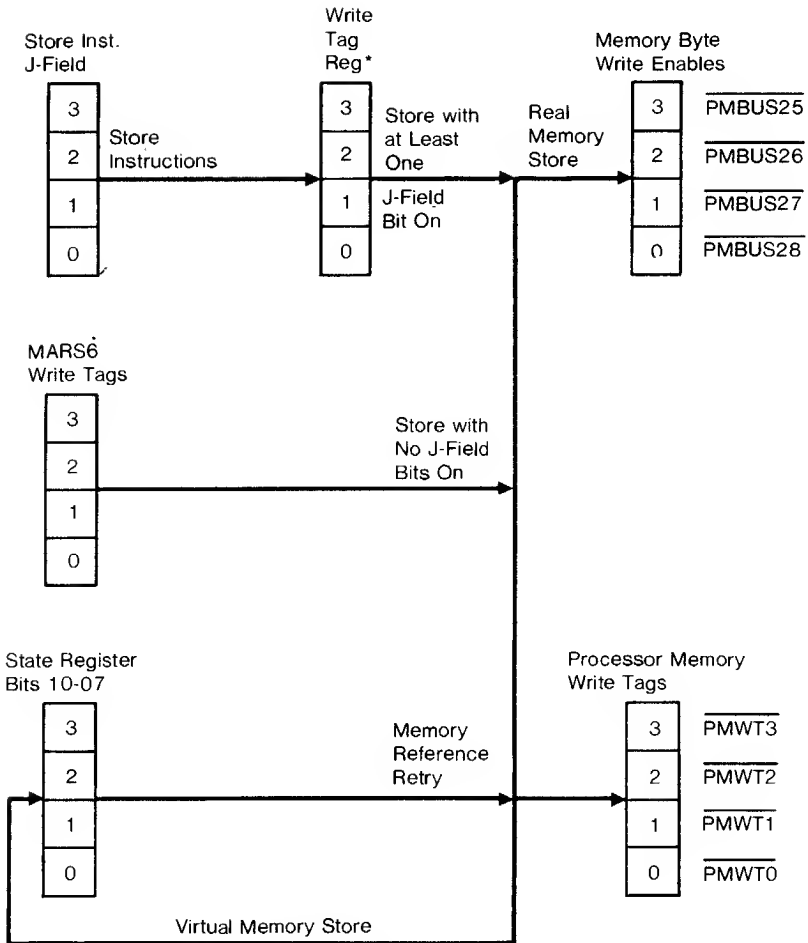
During a Memory Reference Retry instruction, the write tag portion (bits 7-10) of the State Register is used as the source of the write tags. This allows the write tags used during a previously-executed Virtual Store instruction to be used during the retry of that operation.

Figure 4-12 illustrates the transfer paths to and from this internal register.

## SCRATCH PAD ACCESS

The Operand Pointers and the Stack Pointer are used to access the scratch pad. A Transfer Out or a Transfer In via pseudo External





\* This register is an internal register to the CPC and is not user accessible.

GIM4023

Figure 4-12 Write Tag Paths

Registers 32-38 will reference data in the scratch pad and also perform an explicit function on one of the pointers. The scratch pad can also be accessed with the SL and FL instructions.

## ACCESS VIA THE OPERAND POINTERS

A Transfer Out to ERU32 (Operand Data 1) performs a write to the scratch pad word addressed by Operand Pointer #1 if the pointer is in the register mode. A Transfer In from ERU32 performs a read

from the scratch pad word addressed by Operand Pointer #1. Operand Pointer #1 is not altered by either operation.

A Transfer Out to ERU33 (Operand Data 1 Inc) performs a write via OPTR#1 to the scratch pad if the pointer is in the register mode and then increments the pointer to the next word address. A Transfer In from ERU33 performs a read via OPTR#1 from the scratch pad and then increments the pointer to the next word address.

A Transfer Out to ERU34 (Operand Data 1 Dec) performs a write via OPTR#1 to the scratch pad if the pointer is in the register mode and then decrements the pointer to the next word address. A Transfer In from ERU34 performs a read via OPTR#1 from the scratch pad and then decrements the pointer to the next word address.

Operations involving Operand Pointer #2 are identical to those involving Operand Pointer #1. The Operand Data 2 ERUs are substituted for the Operand Data 1 ERUs in the above descriptions. Both OPTR#1 and OPTR#2 are modulo-128. No hardware check is made for the wraparound of a pointer.

Table 4-7 presents a summary of scratch pad access via the pseudo ERU registers.

## ACCESS VIA THE STACK POINTER

A Transfer Out to ERU38 (stack data) performs a write to the operand stack entry addressed by the Stack Pointer and then increments (pushes) the Stack Pointer to the next entry. A Transfer In from ERU38 decrements (pops) the Stack Pointer to the next entry and then performs a read from the operand stack entry addressed by the Stack Pointer.

A Transfer Out to ERUs 32-37 performs the identical operation as a Transfer Out to the stack data ERU if the corresponding operand pointer is in the stack mode. Likewise, a Transfer In from ERUs 32-37 performs the identical operation as a Transfer In from ERU38 if the operand pointer is in the stack mode.

The Stack Pointer operates modulo-32. All addressing in the scratch pad via the Stack Pointer is performed between locations 64 and 95. A reference to the scratch pad using the Stack Pointer presents SPTR 1-5 as the least-significant five bits of the scratch pad word address with (110) asserted on the adjacent three bits.

No hardware check is made for stack overflow or stack underflow conditions.

ERU Address		Register Mode	Stack Mode
Dec.	Hex		
32	20	Operand Data 1	Stack Data
33	21	Operand Data 1 Inc.	Stack Data
34	22	Operand Data 1 Dec.	Stack Data
35	23	Operand Data 2	Stack Data
36	24	Operand Data 2 Inc.	Stack Data
37	25	Operand Data 2 Dec.	Stack Data
38	26	Stack Data	Stack Data

1. Register Mode – Bit 8 of the corresponding Operand Pointer Register (IRU 24) = 0. The Scratch Pad location specified by the contents of Operand Pointer Reg. 1 is accessible through the Scratch Pad Operand Data Registers (20, 21, 22 hex). The Scratch Pad location specified by the contents of Operand Pointer Reg. 2 is accessible through the Scratch Pad Operand Data Registers (23, 24, 25 hex).
2. Stack Mode – Bit 8 of the corresponding Operand Pointer Register (IRU 24) = 1. Scratch Pad access via the Operand Data Registers (20-25 hex) perform the same operation as accessing Scratch Pad via the Stack Data Register (26 hex).
3. The Stack Data Register (26 hex) is unaffected by the value of bit 8 in the Operand Pointer Registers.
4. ERU address locations 32-38 (20-26 hex) represent pseudo registers which allow the user to access Scratch Pad (Scratch Pad locations are determined by Pointer Registers inside the CPC) as ERU transfers.

GIMTE4024A

Table 4-7 Scratch Pad References Via ERU Pseudo Registers

## SCRATCH PAD ACCESS VIA ERU REGISTERS

Scratch Pad accesses via TIE/TOE to ERU locations 20-26 hex are real memory transfers on the PM Bus.  $\overline{\text{EREP}}$  is not asserted (low) as with transfers using other ERU locations. During the execute stage the CPC routes the appropriate pointer to the least-significant bits of the memory word address. Address bits  $\text{PMBUS}_{10-24}$  are forced to the active state (low).

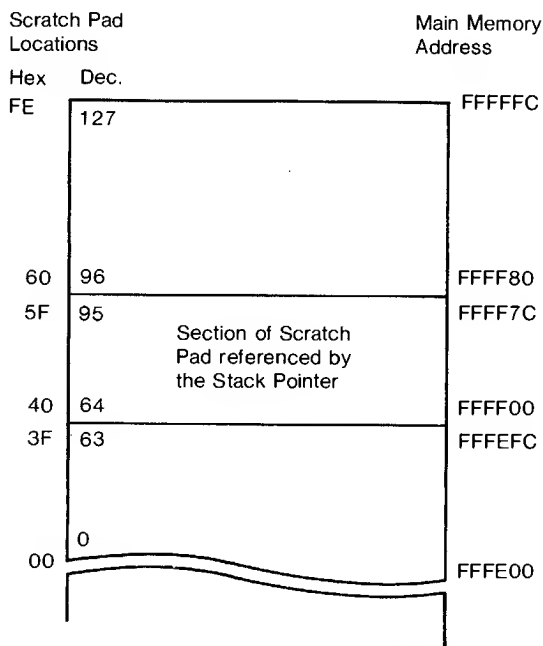
$\text{PMBUS}_{32}$  is forced active (low) by the CPC. This bit can be used to identify a Scratch Pad access.

A TIE from Scratch Pad triggers a real memory read operation. Because of this, an RCV instruction must be used to receive the data.

## SCRATCH PAD ACCESS VIA FL AND SL

The first 64 locations of Scratch Pad (0-63 dec) can be accessed using the Fetch Literal (FL) and Store Literal (SL) instructions. When the CPC executes an FL or SL, a 6-bit address from the instruction is placed on  $\text{PMBUS}_{03-08}$ .  $\text{PMBUS}_{09}$  is negated (high) and the remaining upper address lines  $\text{PMBUS}_{10-24}$  are forced active (low). The transfer is a real memory operation on the PM Bus.

PMBUS32 is forced active (low) by the CPC when either SL or FL is executed. This bit can be used to identify a Scratch Pad access.



- Note:
1. The operand pointers can access the entire scratch pad.
  2. When the Fetch Literal (FL) and Store Literal (SL) instructions are used to reference Scratch Pad they can only access locations 0-63 dec. (00-3F hex). When FL & SL are executed a 6-bit address from the instruction is placed in the least significant 6-bits of the memory word address (PMBUS03-08). PMBUS09 is negated (high) and the remaining address lines are all forced to the active state (low).
  3. During a Scratch Pad reference via the ERU locations the CPC routes the appropriate pointer to the least significant bits of the memory word address. PMBUS10-24 are all forced to an active state (low). This guarantees that the word address that the memory interface receives will be at the top (last physical entries) of memory regardless of the actual main memory size.
  4. PMBUS32 is forced active (low) whenever Scratch Pad is referenced via TIE, TOE, FL, or SL.
  5. Because Scratch Pad accesses are Real Memory operations, a read from Scratch Pad must include the use of the RCV (Receive-Fetched-Data) instruction. Writes to Scratch Pad are effectively one cycle operations to the CPC unless consecutive accesses are at consecutive code locations.

GIM4025A

Figure 4-13 Scratch Pad Portion of Main Memory

## INTERRUPTS/TRAPS

The normal program flow of instructions within the processor may be diverted when certain conditions labeled as program traps or program interrupts occur. Traps are normally considered conditions that, upon detection, must be serviced immediately. Interrupts are normally considered conditions that, upon detection, may be serviced immediately, may be serviced at a later time, or may be masked out altogether.

### INTERRUPT/TRAP RECOGNITION

The trap line ( $\overline{\text{TRAP}}$ ) and interrupt line ( $\overline{\text{INT}}$ ) are external inputs to the processor. When  $\overline{\text{TRAP}}$  is asserted (low) and the trap indicator bit in the Control Array is clear (a trap is not currently being processed), the micro-instruction Control Register is loaded with the trap address (0000 hexadecimal). An immediate jump to the trap routine occurs. The Trap Indicator bit (1) is set, but the Normal Interrupt Enable bit is not altered. The instruction in the Execute stage of the pipeline will execute, but the instructions in the Fetch and the Interpret stages of the pipeline will not be executed. The pipeline will be advanced one cycle and halt without further execution. When the interrupt line is asserted (low) and the Trap Indicator bit in the Control Array is clear and the Normal Interrupt Enable bit is set (an interrupt is not currently being processed), the micro-instruction Control Register is loaded with the interrupt address (0002 hexadecimal). An immediate jump to the interrupt routine occurs. The Normal Interrupt Enable bit is cleared (0). If both an interrupt and trap occur simultaneously, the trap will be serviced first.

Several instructions are non-interruptible. The CPC technical publication (data sheet) should be referenced for a list of these instructions and a description of their effect on interrupt recognition.

Interrupts and traps can be disabled via the Control Array Normal Interrupt Enable and Trap Indicator bits using the TOI, SC, and RC instructions.

### INTERRUPT/TRAP SERVICING

Servicing of traps and interrupts is a function of both hardware and firmware.

Hardware forces the program jump to the trap or interrupt address, clears Normal Interrupt Enable (if an interrupt) or sets Trap Indicator (if a trap), and disables the clocking of the interrupt FIFO.

Firmware must transfer in external interrupt status and test each condition to determine which caused the interrupt or trap. Firmware also determines the priorities of the conditions in the event that multiple interrupts have occurred. To enable further interrupts, firmware must set the Normal Interrupt Enable bit (NIE = bit 10 of Control Array #1) to a 1.

## SAVING THE MACHINE STATE

During some interrupt or trap service routines, it may become necessary to save the state of the processor when circumstances arise that prevent a timely completion of routine (e.g., a page is not resident in main memory during a Dynamic Address Translation [DAT] Fault routine). All internal and external registers pertinent to a recovery of the current task must be saved in the scratch pad or some other reserved section of main memory. The Restore FIFO must be read (three successive Transfers In from IRU08) to obtain the ISU addresses of the three instructions in the pipeline at the time the interrupt or trap was recognized. Control Array #1 and any other status values that were saved at the beginning of the service routine must be relocated to the reserved section of main memory for state saving.

## RESTORING FROM INTERRUPTS/TRAPS

The Restore from Traps and Interrupts (RTI) instruction is used to control the restore sequence. RTI injects the instruction addresses saved in the Restore FIFO into the internal micro-instruction Control Register. By executing three consecutive RTI instructions, the pipeline can be restored to the pre-interrupt/trap state.

To restart the clocking of the Restore FIFO which was disabled when the interrupt/trap was recognized, and to load the skip counter from the Control Array #1, the final RTI instruction must have bit 2 of the K-field set. To clear Trap Indicator and set Normal Interrupt Enable, the second RTI instruction must have bits 1 and 3 of the K-field set. Alternatively, the second RTI instruction may have bit 3 of the K-field clear and bit 1 set, which will clear Trap Indicator and will not affect Normal Interrupt Enable.

If an interrupt or trap is pending at the time the third RTI is executed, the Restore FIFO will not be restarted. The Trap Indicator will be cleared by the RTI, but then immediately set if a trap is pending; Normal Interrupt Enable will be set by the RTI, but then immediately cleared if an interrupt is pending.

## PROCESSOR RESET

The reset signal external to the processor ( $\overline{\text{PMRST}}$ ) initializes the processor during a power-up or system reset sequence.

When  $\overline{\text{PMRST}}$  is asserted, the following actions occur:

1. The Control Register is cleared.
2. BCT, AT, EACA, and NUM in Control Array #1 are all cleared.
3. NIE is cleared and TI is set in Control Array #1.
4. EACBSY in the Indicator Array is cleared.
5. The FIFO is halted after loading the next logical instruction address.
6. All other status and control bits remain unaffected.
7. The RSU is unaffected.
8. The PM bus is negated.

While  $\overline{\text{PMRST}}$  remains asserted, the processor will continuously assert the ISU address of all zeros and skip the execution of all instructions.

If  $\overline{\text{PMRST}}$  is negated (high) and the setup time of the Control Register is met during X1, then the Control Register will increment by one. That cycle and the next cycle will be skip cycles for the Execute stage of the pipeline. During the next cycle, the instruction at ISU location 0000 hexadecimal will have reached the Execute stage and will be executed. The processor at this point is in an entry point to the normal trap routine with the exception that certain Control Array #1 bits have been initialized that would not have been affected by a trap.

## PM BUS ACCESS

All instructions that reference either the main memory or external registers that are not resident in the processor require the CPC to secure the Processor-Memory (PM) Bus before execution of the instruction.

The processor, as a device on the PM Bus, has the lowest priority of all active devices. An active device is one which can initiate a message transfer on the bus. Other active devices gain access to the PM Bus through a request/select protocol. The processor, however, gains access only in the absence of any requests from the other devices, as indicated by the assertion of BAV.

During the Interpret phase of an instruction execution requiring bus availability, the Bus Available (BAV) line is monitored by the CPC. If BAV is inactive, indicating another device has requested the bus for the next cycle, the processor will halt internal operations and stop the pipeline. In the subsequent cycles BAV is tested, and when the bus again becomes available, the instruction executes. The pipeline is then advanced.

The CPC can initiate three types of transfers on the PM Bus:

- External Register Message Transfers
- Real Memory Message Transfers
- Virtual Memory Message Transfers

See Chapter III for a detailed description of PM Bus operations.

## SETUP ASSIST

The NCR/32-000 contains special hardware (Setup Registers 1-5) designed to support emulation of virtual machines. This hardware consists of logic to execute setup instructions and map indicator instructions for the IBM 370 and NCR-VRX/NVM virtual machines. While designed for the above virtual machines, the setup function (setup registers and instructions) of the NCR/32-000 should be considered when emulating other virtual machines.

The setup instructions assist in cracking virtual opcodes, loading the virtual pointers into the Operand Pointers or the Stack Pointer, isolating literal fields in virtual instructions, and creating jump vectors into additional setup routines or into command execution routines.

The map indicator instructions map the processor Indicator Array into the corresponding bits of the virtual indicators.

## SETUP REGISTER APPLICATIONS

The following sections describe Setup Register applications in virtual machine emulation.

### Setup Register #1 Application

The beginning of each virtual command setup routine should contain a setup instruction that loads Setup Register #1 from MARS7. The left or right halfword of the MARS7 Data Register (RSU15), as determined by the instruction type, is transferred to SUR1. This is the portion of the virtual instruction that contains the virtual command code (opcode).



Depending upon the virtual machine type being emulated, the appropriate bits of SUR1 that hold the command format information are decoded. The format determines which of the other bits in SUR1 should be used to create the jump vector to additional setup flows or to the execution flows. The base address portion of the jump address is read from Setup Register #3.

Figure 4-14 shows which setup instructions affect SUR1 and what the decoded format types are for each virtual machine supported. Figures 4-15 through 4-17 show how the jump addresses are formed from SUR1 for each virtual machine supported. Figure 4-18 shows NVM descriptor jump address derivation.

The entire contents of SUR1 can be read by executing a Transfer In from IRU 19. Special instructions are also available which transfer in literal portions of SUR1 right-justified and leading zero-filled:

TSLDC transfers in the contents of SUR1 bits 5-8 to RSU-J bits 1-4 with leading zero-fill.

TSRDC transfers in the contents of SUR1 bits 1-4 to RSU-J bits 1-4 with leading zero-fill.

TSBC transfers in the contents of SUR1 bits 1-8 to RSU-J bits 1-8 with leading zero-fill.

TSB transfers in the contents of SUR1 bits 1-8 to RSU-J bits 1-8 with leading zero-fill for bits 9-16. The left half of RSU-J is undisturbed.

## Setup Register #2 Application

During IBM setup and NVM setup of virtual instructions that are greater than sixteen bits in length, Setup Register #2 is loaded with either the left or the right halfword of the MARS7 Data Register as pointed to by the MARS7 Byte Pointers, using the SETIA and SETNA instructions.

The content of SUR2 is then used to load an Operand Pointer or the Stack Pointer and to transfer a displacement value or offset to RSU-J.

SETIA transfers in the contents of SUR2 bits 1-12 to RSU-J bits 1-12 with leading zero-fill.

SETNA transfers in the contents of SUR2 bits 1-12 to RSU-J bits 1-12 for both the RM and the MM formats. The leading bits of RSU-J are zero-filled.

SETNA transfers in the contents of SUR2 bits 1-16 to RSU-J bits 1-16 for the RI format. Bit 16 of the SUR2 is tested at the time of the transfer. If bit 16 is a zero, the remaining bits of RSU-J are zero-filled. If bit 16 is a one, the remaining bits of RSU-J are one-filled.

The Setup Sign Extension instruction operates similarly to SETNA (with RI format) in that sixteen bits from SUR2 are transferred to RSU-J and bit 16 is tested. However, where SETNA loads SUR2 from the MARS7 Data Register, SUR2 must be preloaded by a Transfer Out prior to execution of the Sign Extension (SETSX) instruction.

### **Setup Register #3 Application**

SUR3 is loaded by a Transfer Out with the base address values used during setup of the particular virtual machine that is being emulated. These base address vectors are then concatenated with the vectors formed during setup to create the required jump addresses.

SUR3 holds a maximum of three base addresses. Bits 9-16 of SUR3 are an 8-bit vector. Bits 5-8 and bits 1-4 are both 4-bit vectors. Listed below are the base vectors used during setup. Figures 4-15 through 4-17 show SUR3 vector fields.

#### **IBM Setup**

- SBA — Setup Base Address
- EBA — Execution Base Address

#### **NVM Setup**

- SBA — Setup Base Address
- IBA — Indirection Base Address
- EBA — Execution Base Address

#### **VRX Setup**

- SBBA — Setup B Base Address
- SABA — Setup A Base Address
- EBA — Execution Base Address

### **Setup Register #4 Application**

SUR4 is loaded by a Transfer Out with the address of the MARS7 Overflow Fetch routine. During IBM or NVM setup, the MARS7 overflow condition is tested, and if it is true, a program jump is executed to the Overflow routine in order to fetch the next virtual instruction.

During NVM emulation, a restriction is placed upon the addresses that can be used for the Overflow routine. Bit 8 must be a zero. This is required because a portion of the SUR4 contents (bits 9-16) is used as a base address during descriptor support.

A descriptor jump is formed by concatenating SUR4 bits 9-16 with a vector formed from the descriptor. Figure 4-18 shows the generation of the descriptor jump address.

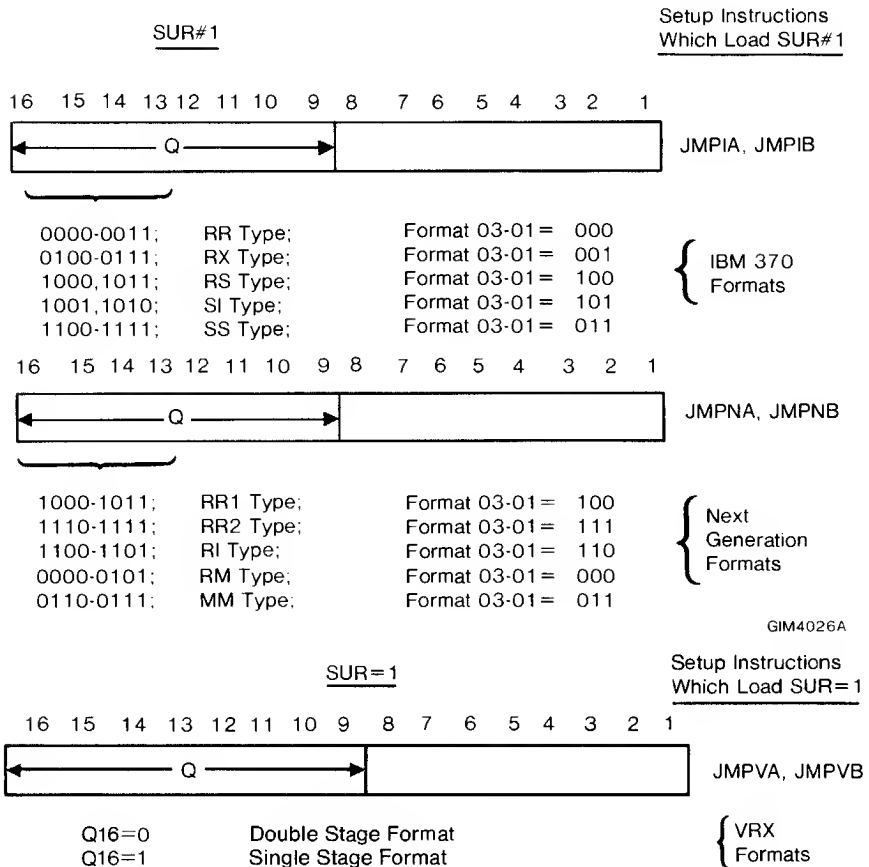
## Setup Register #5 Application

SUR5 is loaded during NVM, IBM, and VRX setup. SUR5 is the Tally Copy Register. The tally field of the virtual instruction is unconditionally loaded into SUR5 bits 1-8 and, subsequently, if the virtual format dictates, SUR5 bits 1-8 are transferred to the Tally Register bits 1-8 by the Load Tally from Setup (LTS) instruction.

During NVM setup, SUR5 bits 1-8 are loaded from the least-significant eight bits of the half-word that is loaded into SUR1 by the JMPNA or JMPNB instruction.

During IBM setup, SUR5 bits 1-8 are loaded from the least-significant eight bits of the half-word that is loaded into SUR1 by the JMPJA or JMPJB instructions.

During VRX setup, SUR5 bits 1-8 are loaded from the most-significant eight bits of the half-word that is loaded into SUR1 by the JMPVB instruction.



GIM4027

Figure 4-14      Formats Decoded from SUR#1 for IBM, NVM, and VRX Setup

CENTRAL PROCESSOR CHIP (CPC)

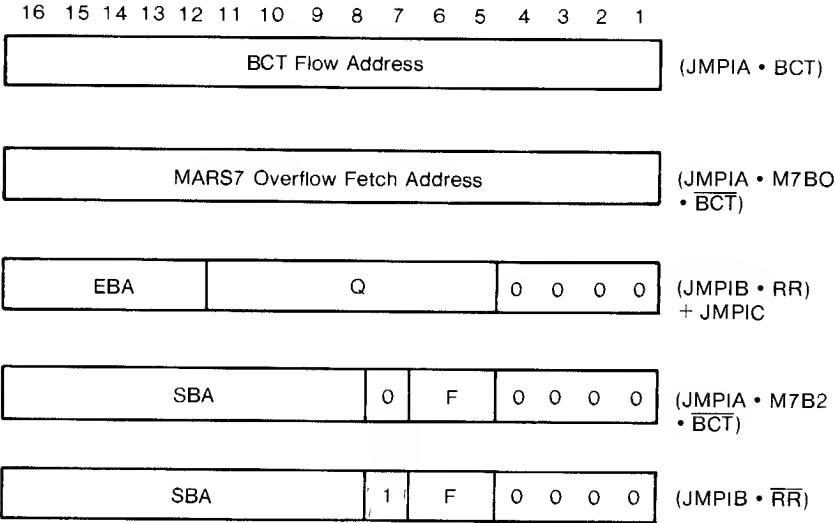
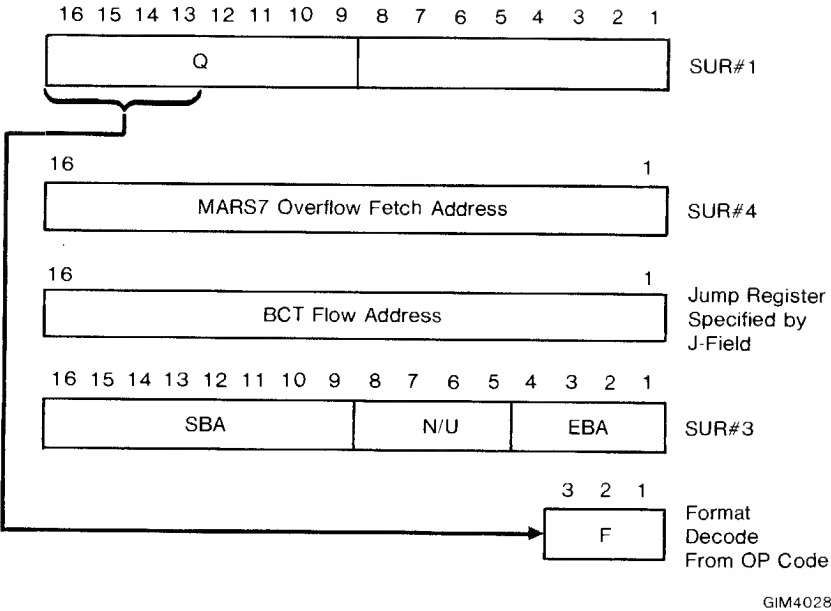
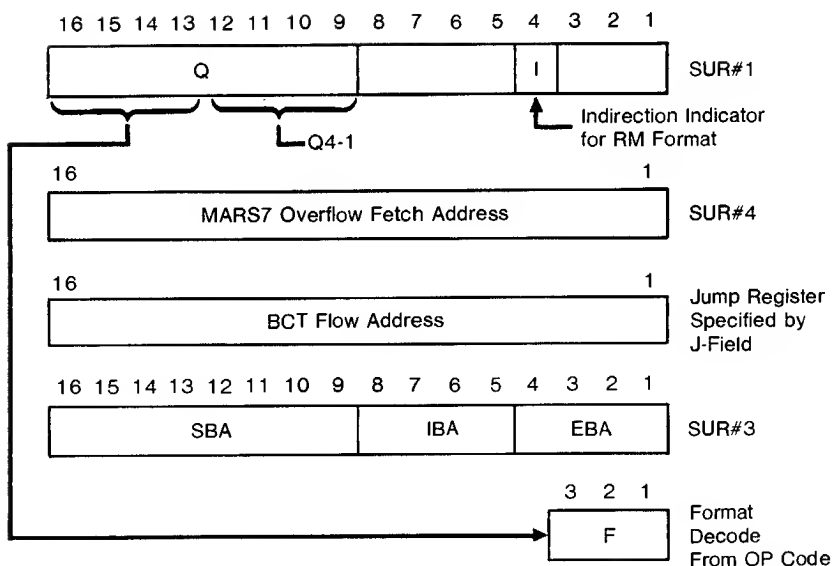
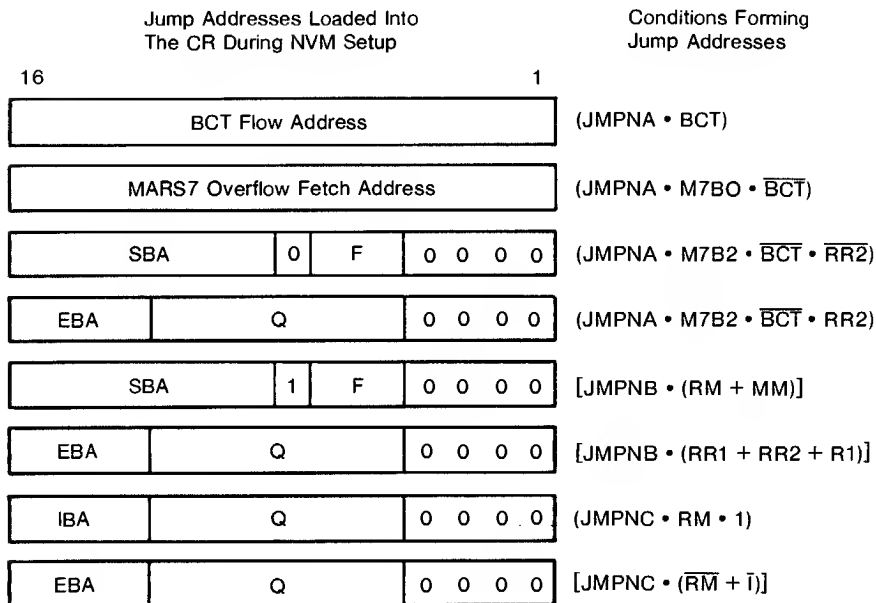


Figure 4-15 IBM Setup Jump Addresses

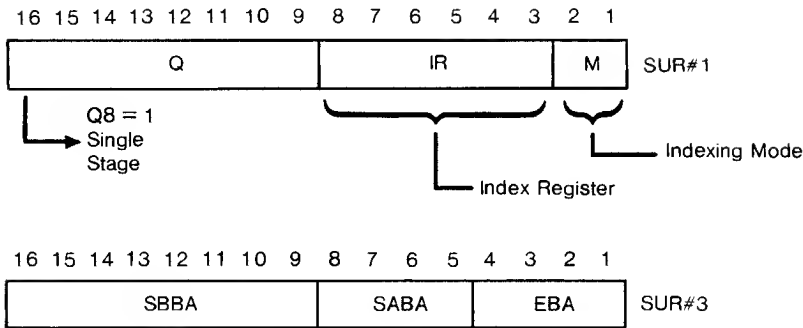


GIM4030



GIM4031

Figure 4-16 NVM Setup Jump Addresses

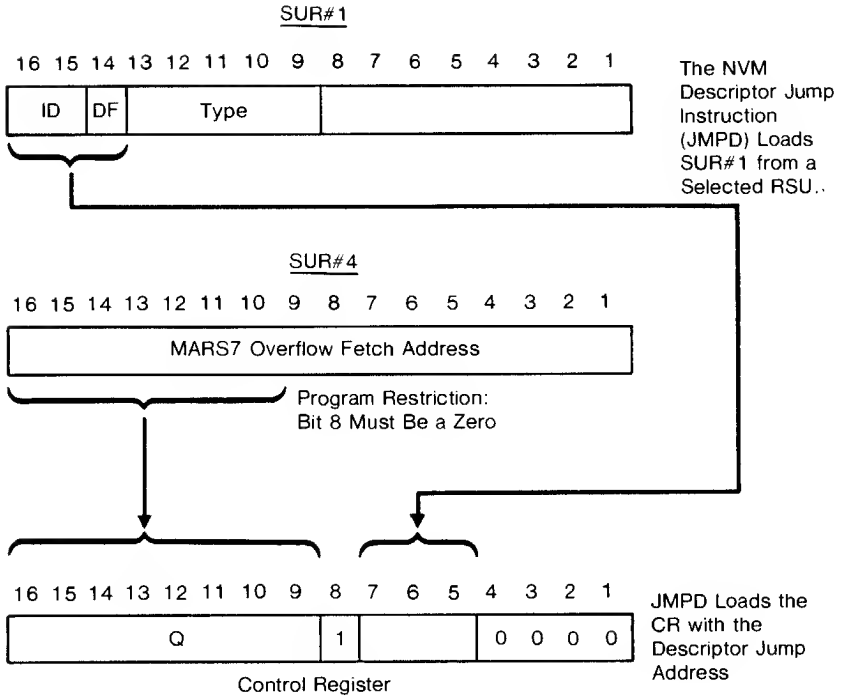


GIM4032

Jump Addresses Loaded into the CR during VRX Setup																Conditions Forming Jump Addresses	
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1		
SABA				0	0	0	Q8	0	0	M	0	0	0	0	0	[JMPVA • (IR = 0 • 63)]	
SABA				0	0	0	Q8	0	1	M	0	0	0	0	0	[JMPVA • (IR = C)]	
SBBA				0	0	0	Q8	1	0	M	0	0	0	0	0	[JMPVA • (IR = 63)]	
SBBA								0	0	M	0	0	0	0	0	[JMPVB • (IR = 0 • 63)]	
SBBA								0	1	M	0	0	0	0	0	[JMPVB • (IR = 0)]	
SBBA								1	0	M	0	0	0	0	0	[JMPVB • (IR = 63)]	
EBA		0	Q7-1								0	0	0	0	0	JMPVC	

GIM4033

Figure 4-17 VRX Setup Jump Addresses



GIM4034

Figure 4-18 NVM Descriptor Jump Addresses

## SCRATCH PAD VIRTUAL MACHINE OPERATION

The last 128 words of main memory are reserved as a scratch pad which can be accessed with the real memory instructions and with the scratch pad external register instructions. Figure 4-19 shows the partitioning of the scratch pad with special assignments for certain virtual machines noted.

The Operand Pointers can be used during virtual command emulation to reference virtual registers located in the scratch pad portion of main memory. The Stack Pointer is used during NVM emulation to reference the operand stack located in the scratch pad.

### Operand Pointer #1

Operand Pointer #1 can be accessed through the normal internal register path via IRU24. During virtual command setup, OPTR#1 bits 1-8 can be loaded as a virtual pointer from a base address and a field in the virtual instruction to facilitate the referencing of the virtual registers located in the scratch pad.

During IBM setup, the JMPPIA and JMPPIB instructions load OPTR#1. OPTR#1 bit 8 is forced to a zero to place the pointer in the register mode. OPTR#1 bits 5-7 are loaded with the base address (110) of the virtual registers. OPTR#1 bits 1-4 are loaded from the contents of SUR1 bits 5-8 (virtual R1 field).

During NVM setup, the JMPNA and JMPNB instructions load OPTR#1. The value (contents of SUR1 bits 5-8) to be loaded into the pointer is tested. If the value is not equal to 15, then OPTR#1 bit 8 is forced to a zero to place the pointer in the register mode, and the remaining bits will be loaded as described. If the value is equal to 15, then OPTR#1 bit 8 is forced to a one to place the pointer in the stack override mode. OPTR#1 bits 5-7 are loaded with the base address (110) of the virtual registers. OPTR#1 bits 1-4 are loaded with the contents of SUR1 bits 5-8 (virtual R1 field).

### Operand Pointer #2

Operand Pointer #2 can be accessed through the normal internal register path via IRU24. During virtual command setup, OPTR#2 bits 1-8 are loaded as a virtual pointer from a base address and a field in the virtual instruction to facilitate the referencing of the Virtual Registers in the scratch pad.

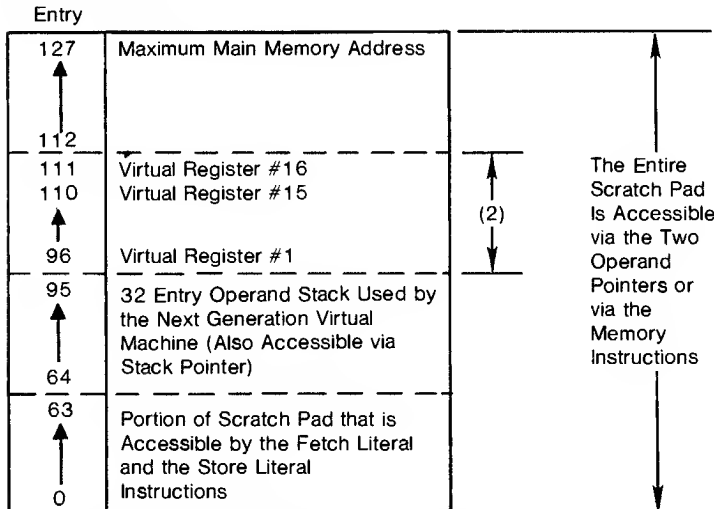
During IBM setup, for opcodes not in the RX format, the JMPPIA and JMPPIB instructions load OPTR#2. OPTR#2 bit 8 is forced to a zero to place the pointer in the register mode. OPTR#2 bits 5-7 are loaded with 110. OPTR#2 bits 1-4 are loaded from the contents of SUR1 bits 1-4 (R2 field).



During IBM setup the SETIA instruction and, for opcodes in the RX format, the JMPA and JMPB instructions also load OPTR#2. OPTR#2 bits 1-4 are loaded from SUR2 bits 13-16 (B field). OPTR#2 bit 5 is loaded with a one if SUR2 bits 13-16 are all zero. Bits 6 and 7 of OPTR#2 are loaded with 11. OPTR#2 bit 8 is forced to a zero to place the pointer in the register mode.

During NVM setup, the JMPNA and JMPNB instructions load OPTR#2. The value (contents of SUR1 bits 1-4) to be loaded into the pointer is tested. If the value is not equal to 15 or the NVM format is the RM type, then OPTR#2 bit 8 is forced to a zero to place the pointer in the register mode and the remaining pointer bits are loaded as described. OPTR#2 bits 5-7 are loaded with 110. If the format is not RM, the OPTR#2 bits 1-4 are loaded from the contents of SUR1 bits 1-4 (virtual R2 field). If the format is RM, the contents of SUR1 bits 1-3 are tested. If they are equal to zero, then a value of 16 (10000) is loaded into OPTR#2 bits 1-5. If they are not equal to zero, then OPTR#2 bit 4 is forced to a zero and the contents of SUR1 bits 1-3 are loaded into OPTR#2 bits 1-3 as the Index Register Pointer.

During NVM setup, the SETNA instruction loads OPTR#2 except for RI formats. OPTR#2 bits 1-4 are loaded from SUR2 bits 13-16 (B-field). OPTR#2 bits 5-8 are loaded as in the JMPNA and JMPNB cases.



1. For IBM or NVM emulation Scratch Pad entry 112 must be pre-loaded with a zero by the firmware.
2. The Operand Pointers are loaded during setup to reference the Virtual Registers.

GIM4035

Figure 4-19 Scratch Pad Partitioning Example for Virtual Machine Emulation

## Stack Pointer

The Stack Pointer can be accessed through the normal internal register path via IRU26. Also, during virtual command setup, the Stack Pointer can be selected for subsequent use by a field in the NVM virtual instruction.

During NVM setup, if the value tested from SUR1 or from SUR2 (excluding the Index Register case for the RM format) is equal to 15, then the Stack Pointer is selected. Bit 8 in OPTR#1 or OPTR#2, as appropriate, is set to a one, overriding the use of that pointer on subsequent external register references. Instead, the Stack Pointer is used (transparently to the firmware) to access the Operand Stack.

## MAP INDICATOR LOGIC

The most frequently modified virtual indicators are provided special CPC logic to map the corresponding bits in the processor Indicator Array into the virtual indicator counterparts. Each of the three virtual machines explicitly supported has a map indicator instruction associated with it.

The J-field and the K-field of the instruction act as enables for the mapping process. Thus, if during emulation of a virtual command only the L, E, and G flags should be affected, then the L, E, and G enables in the map indicator instruction are set. The other flags or indicators in the Indicators Array will not be modified.

Virtual indicators that must be modified during emulation but are not supported by mapping can be changed via the Transfer Out instruction (TOI to IRU17).

## PROGRAMMING CONSIDERATIONS

This section describes programming considerations and restrictions when using the NCR/32-000 processor in a system.

## FIELD OPERANDS

Field strings are from 1 to (64 K-1) bytes in length. The fields are located in the MSU (Main Memory). The processor works on one, two, or three fields at a time. A field is specified by the contents of a MARS register and the contents of the Tally Register. The MARS registers are specified by the instruction J-field and K-field. The store MARS is always MARS6, implied by the instruction. All fields must be of equal length.

The MARS operation is specified by the instruction operation code. Fields are processed one byte at a time under CPC logic control. The field instructions do not move in the pipeline until a word boundary is crossed or the Tally Register equals zero (except for the CFU instruction, which also moves in the pipeline when either bits 1 or 3 are set in the Indicator Array in response to a non-equal comparison of two fields). The data is transferred between the MSU and RSU (four bytes at a time) under firmware control. While in the RSU, the bytes are addressed by the two low-order bits of the corresponding MARS register. Thus, the instruction selects the RSU word and the MARS selects the byte within the word. The tally is decremented as each byte is processed.

When the tally equals zero the field operation is complete, the pipeline is advanced to the next instruction, and the firmware sends a partial store to the MSU if required.

If the tally equals 0 when the field command is entered, then the specified transfer is not executed.

The MARS Data Registers to be used in a field instruction must be initialized before entering the field operation. For arithmetic field instructions, the carry indicator must be pre-set to the proper value.

The Carry indicator is the only indicator that responds (set/clear) through the individual cycles of the field instructions. The Carry (I4) must be initialized by firmware prior to the first execution of an AF, APDF, AUDF, SF, SPDF, or SUDF instruction. The carry is chained automatically throughout the subsequent execution cycles. I4 is initialized to a zero (with the RIZ instruction) for AF, APDF, or AUDF. I4 is initialized to a one (with the SCO instruction) for SF, SPDF, or SUDF.

## **SINGLE FIELD OPERAND INSTRUCTIONS**

The byte to field, halfword to field, field to byte, and field to halfword instructions are all single field operand instructions. Only one of the instruction operands is a field operand. These instructions execute in a single cycle.

The following is a firmware flow using a single field operand instruction:



GIM4036

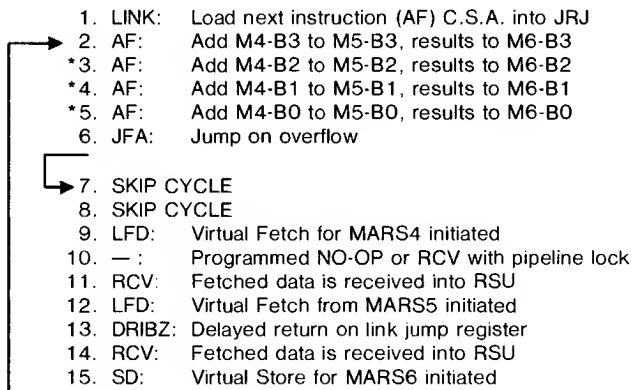
Figure 4-20 Single Field Operand Routine

## Multiple Field Operand Instructions

The transfer field instructions, the boolean field instructions, and the arithmetic field instructions are multiple field operand (two and three field) instructions. These instructions execute in from one to four cycles depending upon whether a MARS byte pointer crosses the word boundary or the Tally Register decrements to zero.

These instructions hold in the Execute stage of the pipeline until an exit condition occurs.

The following is a firmware flow using a multiple field operand instruction where all fields are aligned:



\* AF remains in the execute stage until an exit condition occurs (MARS overflow condition or Tally = 0).

GIM4037A

Figure 4-21 Multiple Field Operand Routine

## FETCHING FROM ISU

The processor instruction set does not support an explicit instruction which allows fetching of data (tables, etc.) from the ISU. However, a sequence exists which effectively performs the same function. This sequence depends upon the use of a 2-word instruction beginning as the second instruction following an unconditional delayed jump. Ordinarily, this is considered a violation of a restriction involving delayed jumps since the literal used by the 2-word instruction will not follow from the coded flow. In realizing what the CPC does in this circumstance, though, the restriction can be overlooked and used to effect a fetch from control store.

The sequence to produce a fetch from control store is:

1. Delayed jump
2. Delayed jump
3. LRH or LRHC
4. X

The delayed jump in line 1 should specify the desired control store address to be fetched from (e.g., DJOR, DRIBZ, etc.). The delayed jump in line 2 should return the program flow to the desired instruction code (could be instruction 4). The LRH or LRHC instruction, when it executes, will pick up the instruction to be used as a literal that is in the pipeline Interpret stage while the LRH or

LRHC is in the Execute stage. With the above sequence, that instruction (literal) will be the instruction referenced by the delayed jump in line 1.

Thus, by varying the address specified by the first delayed jump, this sequence can be used to fetch any number of words from the ISU.

## DELAYED JUMPS

An immediate program branch (e.g., immediate jump) voids the 3-stage pipeline. The delayed jump increases efficiency by allowing the next two instructions already loaded in the pipeline to execute. The efficient programmer is able to maximize performance by using delayed jump or return instructions.

## LOCK ON FETCH

The Receive Fetched Data (RCV) instruction should be executed when the data resulting from a fetch operation is available from memory. The processor pipeline halts until the Data Input Enable ( $\overline{DIE}$ ) signal is recognized by the processor.  $\overline{DIE}$  is asserted one cycle before the data is asserted on the PM bus. At this time, the pipeline is advanced so that RCV execution coincides with data assertion on the PM Bus.

This mechanism for holding the pipeline is identical to that for Bus Availability (BAV) monitoring as described in the "Bus Access" section. The only difference is that the processor cannot be interrupted or trapped while the PM bus is unavailable, but can be interrupted while waiting for memory data.

## STORE OPERATIONS

Store to memory operations are initiated by the explicit Store type primitive instructions (SR,S,SA,SD,SL,MRR) and by the Implicit Scratch Pad ERU references (TOE to ERUs 32-38).

All cycles after the first cycle, during which the Store instruction is executed, are offline to the Processor. Unless the Processor attempts to execute another memory operation or some instruction that accesses the PM Bus prior to the completion of the actual write to memory of the Store data, Store operations can be considered to be one cycle operations.

The number of offline cycles executed during Store operations is a function of whether address translation is performed, whether the Store is a full word Store or a partial word Store, and memory access time.

## FETCH OPERATIONS

Fetch from Memory operations are initiated by the explicit Fetch type primitive instructions (FR,F,LFA,LFD,LFAL,FL, MRR) and by the implicit Scratch Pad ERU references (TIE from ERUs 32-38).

Fetches, unlike Stores, require two instructions to be executed by the Processor. The first, as for Stores, triggers the memory operation. The second, which is unique to Fetch operations, loads the accessed Memory data into an RSU. This second instruction is the Receive Fetched Data (RCV) instruction.

The minimal two cycle Fetch sequence consists of a Fetch Instruction followed by an RCV instruction. The variations from this sequence depend on whether Address Translation is performed and memory access time. These other cases can be handled in one of two ways.

Alternative #1: a simple macro is used by Firmware whenever a Fetch sequence is required. This macro corresponds to the minimal Fetch sequence. Hardware resolves any variations in Fetch timing by halting the Processor pipeline prior to executing the RCV until the fetched data is available on the PM Bus.

Alternative #2: Firmware recognizes the state of the AT control bit at all times and the particular memory access time for the memory components in any given machine. The RCV, which is a dynamic instruction in that whatever is currently on the PM Bus is loaded into RSU, is executed, then, according to the variable factors.

Alternative #2 presents a performance advantage in that any Processor cycles that occur between the Fetch instruction and the RCV instruction can be utilized provided they do not reference the PM Bus. Alternative #1 presents a Firmware management advantage in that each sequence need not be specially tailored and multiple sets of firmware need not be supported for different component access times.

## PROGRAMMING RESTRICTIONS

The following are programming restrictions for the NCR/32-000 processor. Included are CPC restrictions which apply to CPC-based systems which utilize the NCR/32-010 Address Translation Chip.

1. The Receive Fetched Data (RCV) instruction used in the program flow of all fetch operations must be executed when the data to be received is asserted on the PM Bus. RCV must be either properly located in the fetch firmware flow, or must be executed in response to DIE assertion to ensure proper data reception.

2. Virtual memory store instructions capable of triggering DAT interrupts should not be followed by a virtual memory instruction. At least one interruptible instruction should be included between those virtual memory instructions.
3. The instructions immediately preceding any instruction which uses the MARS byte pointers (bits 1 and 2 of a MARS address register), other than an increment or decrement in a field or setup instruction, must not change the value of the pointers. Those instructions (LTS, LTRC, JMPVC, and TOE to IRU12) which load the Tally Register may not immediately precede any field instruction which uses the tally.
4. The second instruction following a delayed jump instruction should not be a two-word instruction (an instruction requiring an "L" field trailing literal).
5. The skip instructions cannot be used to skip 2-word instructions.
6. The restore from interrupts/traps sequence must include three successive RTI instructions to restore the pipeline to the pre-trap/interrupt state.
7. A program sequence which uses field instructions having more than one active MARS unit must include a load link address instruction (LINK) to establish the re-entry address from the overflow routines into a jump register.
8. The PS, PSM, and TVA functions executed via the TOE instruction must not be followed by an instruction which references the PM bus.
9. Scratch pad entry 112 must be cleared to a zero value by firmware during NVM emulation for use as the zero value which is read when Index Register 0 is referenced.
10. If the virtual address for a fetch operation that causes a memory error trap must be recovered, then the instruction immediately following the fetch sequence may not be a virtual memory instruction (which will alter the previous virtual address).
11. All virtual store or fetch instructions which have AT set must use an odd-numbered RSU as the data source/destination RSU.
12. Except during a setup sequence other than SETIA or SETNA, when the Operand Pointers and Stack Pointer are loaded via special hardware, the pointers can not be loaded in one processor cycle (instruction) and then used to reference the scratch pad data in the next cycle.
13. The RSU specified by the J-field in the SETIA, SETNA, TSBC, TSLDC, and TSRDC instructions or the K-field in the SETSX instruction may not be used as a source RSU in the instruction immediately following one of these instructions.
14. A TOE to the TOD Register (ERU44) must not change bits



1-10. Immediately preceding the point where the TOE is to be executed, a TIE must be executed from the TOD, and bits 1-10 extracted and concatenated with the value to be loaded into bits 11-32 of the TOD.

## CHAPTER V ADDRESS TRANSLATION CHIP (ATC)

### CONTENTS

ATC FUNCTIONAL DESCRIPTION .....	5-7
Memory Operations .....	5-7
Virtual Memory Operations (Address Translation) .....	5-8
Real Memory Operations .....	5-9
Memory Refresh Operation .....	5-9
Error Check/Correction and	
Syndrome Bit Generation (ECC) .....	5-10
ECC Generation During Memory Store .....	5-11
Error Check and Correction During Memory Fetch .....	5-11
Multi-Work Fetch/Correction .....	5-12
External Register Unit Operations .....	5-12
Time Of Day/Interval Monitoring .....	5-13
Virtual Address Monitoring .....	5-13
Breakpoint (Fetch for Execute) Monitor — CA2 .....	5-14
Address Monitor (Stores) — CA3 .....	5-14
Fetch for Execute Monitor (Stores) — CA3,4 .....	5-14
Trace (Store) Monitor — CA8 .....	5-14
 DYNAMIC ADDRESS TRANSLATION UNIT .....	 5-14
Overview .....	5-18
Operation .....	5-18
Translation .....	5-18
Memory Protection .....	5-19
Invalid Register (IR) .....	5-20
Changed Page (CP) .....	5-20
Register Referenced (RR) .....	5-20
Protection .....	5-21
Page Frame Number .....	5-21
Virtual Page Number .....	5-22
Interrupts .....	5-22
 EXTERNAL REGISTER DEFINITIONS .....	 5-22
Special Purpose ERU .....	5-23
Control Array #2 .....	5-23
Control Array Definition .....	5-24

# **CHAPTER V** **ADDRESS TRANSLATION CHIP (ATC)**

## **CONTENTS (CONTINUED)**

Interrupt/Trap Array .....	5-26
Trap and Interrupt Operation .....	5-26
Trap and Interrupt Definition .....	5-27
Interrupt Mask Register (IMR) .....	5-30
Interval Timer/Monitor Register (ITMR) .....	5-30
Time-Of-Day Register/Counter (TOD) .....	5-30
Address Monitor Register (AMR) .....	5-31
Bus Interrupt Register (BIN) .....	5-31
Syndrome Register (SR) .....	5-31
Memory Data/Processor Data Register (MD/PD) .....	5-32
Virtual Operation ERUs .....	5-32
Virtual Address Register (VAR) .....	5-32
Real Address Register (RAR) .....	5-33
Descriptor Data Register (DDR) .....	5-34
Associative Memory and Page Descriptor Registers .....	5-35
 ASSOCIATIVE MEMORY COMMANDS .....	 5-35
Write Page Size (WPS) .....	5-36
Read Page Size (RPS) .....	5-36
Invalidate Associative Memory (IAM) .....	5-36
Enable and Set Page Frame (ESPF) .....	5-36
Restriction .....	5-37
Write Virtual Page (WVP) .....	5-37
Read Page Frame (RPF) .....	5-38
Write and Set Page Frame (WSPF) .....	5-38
Restriction .....	5-39
Clear Associative Memory (CAM) .....	5-39
Purge Selective (PS) .....	5-39
Restrictions .....	5-40
Write Purge Mask (WPM) .....	5-40
Read Purge Mask (RPM) .....	5-41
Purge Selective with Mask (PSM) .....	5-41
Restrictions .....	5-42
Read Virtual Address (RVA) .....	5-42
Read Real Address (RRA) .....	5-42
Translate Virtual Address (TVA) .....	5-44
Restrictions .....	5-45

## CHAPTER V

### ADDRESS TRANSLATION CHIP (ATC)

## CONTENTS (CONTINUED)

ATC STATE OPERATION .....	5-45
State Flow .....	5-47
SPECIAL ATC CONSIDERATIONS .....	5-47
PM Bus Contention .....	5-47
Refresh .....	5-47
Time-Of-Day .....	5-48
Bus Interrupt Register Interrupts .....	5-48
Associative Memory Command Sequencing .....	5-48
Monitor Operations .....	5-48
ECC Disable .....	5-48
ECC Generate/Syndrome Register .....	5-49
Real Address Register Byte/Descriptor Data Register .....	5-49
24/32 Bit Operations .....	5-49
Associative Memory Results .....	5-49
TIMING CYCLE DESCRIPTIONS .....	5-49
Real Memory Operations .....	5-49
Real Full Store .....	5-50
Real Partial Store .....	5-50
Real Fetch .....	5-50
Virtual Memory Operations .....	5-51
Virtual Full Store (CA9=0) .....	5-51
Virtual Partial Store (CA9=0) .....	5-52
Virtual Fetch (CA9=0) .....	5-52
Virtual Full Store (CA9=1) .....	5-53
Virtual Partial Store (CA9=1) .....	5-53
Virtual Fetch (CA9=1) .....	5-53
Refresh Operation .....	5-54

## CHAPTER V ADDRESS TRANSLATION CHIP (ATC)

The NCR/32-010 Address Translation Chip (ATC) is an NMOS, 32-bit memory management unit that provides system memory management, data error detection/correction, memory refresh, supervisor/user isolation using four-level memory access protection, and virtual address translation for memory fetches, full stores, and partial stores. A functional block diagram of the ATC is shown in Figure 5-1.

The ATC is packaged in a 68-pin leadless chip carrier (Figure 5-2), and features the following:

- NCR/32-000 Compatibility
- Memory Management with four-level memory access protection
- Three types of Memory Operations
  - Virtual
  - Real
  - Refresh (Main Memory)
- Multi-Word Fetch/Correction
- Error Check and Correction, and Syndrome Bit Generation
- Time-of-Day Counter and Time Interval Monitoring
- Virtual Address Monitoring for Breakpoint and Trace Functions

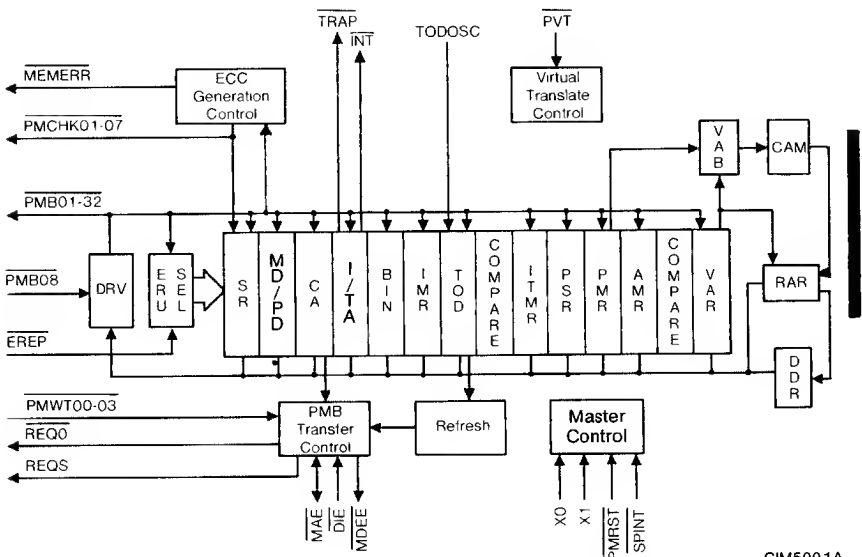
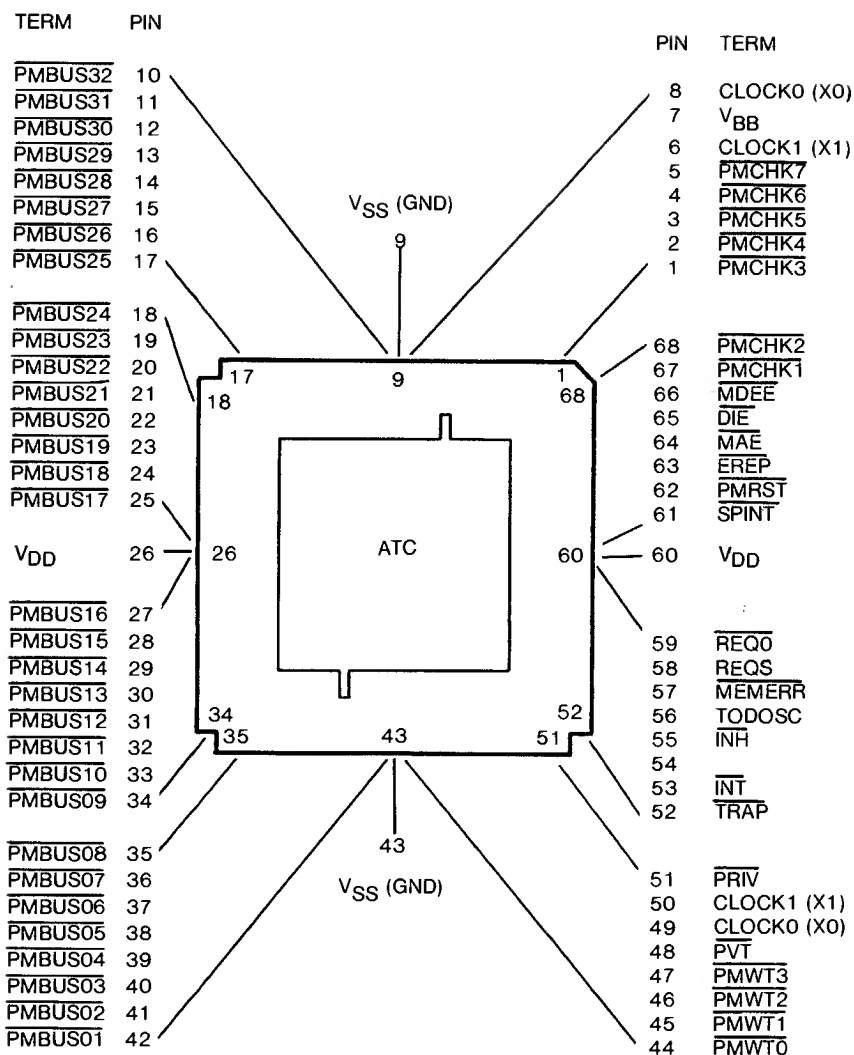


Figure 5-1      ATC Functional Block Diagram

# Address Translation Chip (ATC)



GIM5002B

Figure 5-2 ATC Pin Assignment—Bottom View (Opposite From Lid)

Table 5-1 contains a brief description of the ATC input/output signals, and Table 5-2 gives a summary of the signals.

PIN #	Symbol	Description
10-25 27-42	PMBUS32-01	Processor Memory Bus—These multiplexed bus lines are used to transfer information between devices. In general, address information is transferred to a destination device during X0, and data is transferred to/from a destination device during X1. However, all address and data transfers between the ATC and memory occur during X0.
8, 49	X0	Phase 0 input clock.
6, 50	X1	Phase 1 input clock.
62	$\overline{\text{PMRST}}$	PM Bus Reset—This input, when asserted, holds the ATC in a reset state.
61	$\overline{\text{SPINT}}$	Special Interrupt—This input is sampled by the ATC during X1. The ATC responds to an asserted $\overline{\text{SPINT}}$ by setting bit 12 (Special Interrupt) of the I/T Array and asserting $\overline{\text{INT}}$ if the interrupt is not masked in the Interrupt Mask Register. $\overline{\text{SPINT}}$ is normally tied to the system low-voltage/power-fail detection circuit.
63	$\overline{\text{EREP}}$	External Register Enable/Permit—When this input is asserted during X0, the ATC decodes the External Register Unit number from the address lines to determine if one of the ATC External Register Units (registers) is being addressed. Descriptions of the ATC ERUs are given in this chapter.
64	$\overline{\text{MAE}}$	<p>Memory Address Enable—This input, when asserted, indicates that a device has initiated a real memory operation. If the operation is a fetch, the ATC reads and checks the data for errors during the X0 following the assertion of <math>\overline{\text{MAE}}</math>, and asserts the checked data on the PM Bus the following X1 for reading by the requesting device.</p> <p>The ATC is forced to execute multiple fetches/corrections by the assertion of <math>\overline{\text{MAE}}</math> and <math>\overline{\text{DIE}}</math> during subsequent X0 clocks, and remains in the fetch/correct state until <math>\overline{\text{MAE}}</math> assertion during X0 ceases.</p> <p>The ATC asserts <math>\overline{\text{MAE}}</math> when it asserts a real address on the PM Bus after address translation in response to a virtual memory message; that is, if CA9 (Virtual Equals Real) in the Control Array is clear, the ATC asserts <math>\overline{\text{MAE}}</math> and a real address on the PM Bus during X0 if PVT was asserted the previous X0, indicating a virtual memory operation.</p>

GIMTE5003A-1

Table 5-1 ATC Pin Description

## Address Translation Chip (ATC)

PIN #	Symbol	Description
48	$\overline{PVT}$	Processor Virtual Transfer—The ATC initiates a virtual address translation when this input signal is asserted low during X0 if bit 9 (Virtual Equals Real) is clear in the Control Array. If bit 9 is set when $\overline{PVT}$ is asserted during X0, the ATC initiates a real memory access.
44-47	$\overline{PMWT0-3}$	Processor Memory Write Tags 0-3—The ATC monitors these inputs during virtual memory operations to determine which bytes in the addressed word are to be written. These signals are equivalent to the Byte Write Enable signals in real memory messages. If all of the write tags are negated, a fetch is indicated. If one or more are asserted, corresponding byte(s) are written. $\overline{PMWT0}$ corresponds to byte 0, $\overline{PMWT3}$ corresponds to byte 3, etc.
67, 68 1-5	$\overline{PMCHK7-1}$	PM Check 7-1—These bidirectional lines are used for syndrome bit (ECC) transfer between the ATC and memory. These signals are asserted by the ATC during store operations, and by the memory during fetch operations.
65	$\overline{DIE}$	Data Input Enable—This signal is asserted by the system Memory Interface during X0 and X1 to indicate that memory data is to be asserted on the PM Bus as part of a fetch, partial store, or refresh operation; or that the Memory Interface is ready to accept data.
66	$\overline{MDEE}$	Memory Data/Enable Error—The ATC asserts this signal during the X1 clock that it detects an uncorrectable (multiple bit) memory error during a fetch or partial store operation.
57	$\overline{MEMERR}$	Memory Error—The ATC asserts this signal during X1 in response to a correctable (single bit) or non-correctable (multiple bit) error detected during a partial store, fetch, or refresh operation.
59	$\overline{REQ0}$	Request 0—The ATC asserts $\overline{REQ0}$ to gain access to the PM Bus the following PM Bus cycle. $\overline{REQ0}$ is the highest priority system request. When the ATC asserts $\overline{REQ0}$ , it takes control of the PM Bus the following cycle without further handshaking. The system bus arbitration logic must force other devices off the PM Bus at this time. The ATC may continue asserting $\overline{REQ0}$ for several consecutive cycles to complete an operation.
58	REQS	Special Request—The ATC asserts this line high to give the CPC immediate access to the PM Bus. While REQS remains asserted, the system bus arbitration logic must ignore all other bus requests, and de-select all other devices.

Table 5-1 ATC Pin Description (Continued)

GIMTE5003A-2



PIN #	Symbol	Description
		Uncorrectable memory errors require immediate CPC intervention before other data transfers occur. When the ATC detects an uncorrectable data error, it asserts REQS until the CPC has serviced the error condition, indicated by CPC clearing of the trap bit in the ATC I/T Array.
		Three ATC associative commands force the ATC to assert REQS: the Purge Selective (PS) command, the Purge Selective With Mask (PSM) command, and the Translate Virtual Address (TVA) command. The ATC asserts REQS for one cycle in response to these commands.
		ATC memory refresh operation is inhibited while REQS is asserted.
53	$\overline{\text{INT}}$	Interrupt—The ATC asserts $\overline{\text{INT}}$ when any bit in the I/T Array other than a trap bit is set. The ATC negates $\overline{\text{INT}}$ when the bit forcing the interrupt is cleared, if no other interrupts are pending. $\overline{\text{INT}}$ assertion is inhibited during X1.
52	$\overline{\text{TRAP}}$	Trap—The ATC asserts $\overline{\text{TRAP}}$ when any trap bit in the I/T Array is set. The ATC negates $\overline{\text{TRAP}}$ when the bit forcing the trap is cleared, if no other traps are pending. Traps cannot be masked. $\overline{\text{TRAP}}$ assertion is inhibited during X1.
56	TODOSC	Time Of Day Oscillator—This input is driven by a free-running 250 KHz external oscillator, and is used to drive the TOD (Time Of Day) Counter/Register in the ATC. The frequency of this input affects the memory refresh rate.
55	$\overline{\text{INH}}$	Inhibit—This input, when asserted low, inhibits ATC refresh operation to allow system expansion.
51	$\overline{\text{PRIV}}$	Privilege Mode— $\overline{\text{PRIV}}$ is an input used to set/reset the Privilege Flag, bit 1, in the Control Array. $\overline{\text{PRIV}}$ must first be asserted during X0. If $\overline{\text{PRIV}}$ is asserted the following X1, the Privilege Flag is set; if $\overline{\text{PRIV}}$ is negated the following X1, the Privilege Flag is cleared.
7	$\text{V}_{\text{BB}}$	Negative voltage supply.

GIMTE5003A-3

Table 5-1 ATC Pin Description (Continued)

## Address Translation Chip (ATC)

PIN #	Symbol	Description
9, 43	V <sub>SS</sub>	Ground
26, 60	V <sub>DD</sub>	Positive Voltage Supply
54	NC	No Connection (not used)

Barred terms are active low.

GIMTE5003A-4

Table 5-1 ATC Pin Description (Continued)

Signal Name	Pin #	Symbol	Input/ Output	Active State	Drive
Processor-Memory Bus	10-25 27-42	$\overline{\text{PMBUS32-01}}$	Input/ Output	Low	Open Drain
Clock 0	8, 49	X0	Input	High	Input
Clock 1	6, 50	X1	Input	High	Input
PM Bus Reset	62	$\overline{\text{PMRST}}$	Input	Low	Input
Special Interrupt	61	$\overline{\text{SPINT}}$	Input	Low	Input
External Reg. Enable/Permit	63	$\overline{\text{EREP}}$	Input/ Output	Low	Open Drain
Memory Address Enable	64	$\overline{\text{MAE}}$	Input/ Output	Low	Open Drain
Processor Virtual Transfer	48	$\overline{\text{PVT}}$	Input	Low	Input
Processor Write Tags	44-47	$\overline{\text{PMWT0-3}}$	Input	Low	Input
Processor Memory Check Bits	5-1 68, 67	$\overline{\text{PMCHK7-1}}$	Input/ Output	Low	Open Drain
Data Input Enable	65	$\overline{\text{DIE}}$	Input	Low	Input
Memory Data Enable/Error	66	$\overline{\text{MDEE}}$	Input/ Output	Low	Open Drain
Memory Error	57	$\overline{\text{MEMERR}}$	Output	Low	Open Drain
PM Bus Request "0"	59	$\overline{\text{REQ0}}$	Output	Low	Open Drain

Table 5-2 Pin Assignment Summary

Signal Name	Pin #	Symbol	Input/ Output	Active State	Drive
Special Request	58	REQS	Output	High	Push-Pull
Interrupt	53	$\overline{\text{INT}}$	Output	Low	Open Drain
Trap	52	$\overline{\text{TRAP}}$	Output	Low	Open Drain
Time-of-Day Osc.	56	TODOSC	Input	High	Input
Inhibit	55	$\overline{\text{INH}}$	Input	Low	Input
Privilege Mode	51	$\overline{\text{PRIV}}$	Input	Low	Input
Power Supply (Pos)	26, 60	$V_{DD}$	Input	—	—
Power Supply (Neg)	7	$V_{BB}$	Input	—	—
Ground	9, 43	$V_{SS}$	Input	—	—

GIMTE5004A2

Table 5-2 ATC Pin Assignment Summary (Continued)

## ATC FUNCTIONAL DESCRIPTION

This section gives descriptions of the following ATC functions: memory operations, error check/correction, External Register Unit operations, time of day/interval monitoring, and virtual address monitoring. The rest of the chapter is devoted to detailed descriptions of the ATC Dynamic Address Translation Unit, External Register Units, and Associative Memory commands necessary for implementation of the ATC in a system.

### MEMORY OPERATIONS

The ATC has an internal state machine to sequence memory operations (see ATC State Operation). Chapter III describes memory operations and PM Bus protocols.

## Virtual Memory Operations (Address Translation)

The ATC can operate on either 24 or 32 bit virtual addresses.

The ATC determines that a virtual operation is being initiated when the  $\overline{PVT}$  (Processor Virtual Transfer) line is asserted during X0, at which time a virtual address is on the PM Bus. The ATC strobes the address into its Virtual Address Register Buffer and starts the translation process. If the translation process is not successful, the ATC generates a DAT (Dynamic Address Translation) No Match Interrupt. If the translation process is successful and protection check is enabled, the ATC protection check bits associated with the virtual address are checked according to the operation indicated by PM Bus bits 2 and 1 as follows:

### BIT 02 01

0	0	Virtual Fetch
0	1	Virtual Store
1	0	Virtual Fetch for Linkage
1	1	Virtual Fetch for Execution

NOTE: These signals are logical values (i.e.  $\overline{PMBUS02} = 0$  is a logical value and represents an electrical high value on the PM Bus).

If there is an access violation, the ATC generates a DAT Access Violation Interrupt.

Assuming a successful translation, the real 24 bit address is in a register used for holding real addresses, called the Real Address Register, by the end of X1.

During the next X0 the ATC asserts  $\overline{MAE}$  (Memory Address Enable) and asserts the real address on the PM bus.

During the X0 clock that  $\overline{PVT}$  is asserted, the ATC asserts the  $\overline{REQ0}$  (PM Bus Request 0—highest priority request) to secure access to the PM Bus in the next cycle, and examines the  $\overline{PMWT0-3}$  (Processor-Memory Write Tags 0 thru 3) lines. If  $\overline{PMWT0-3}$  are all asserted, the ATC enters a full store sequence. If some but not all PMWTs are asserted, the ATC enters a partial store sequence (i.e., a read-modify-write). If all PMWTs are negated the ATC enters a fetch sequence. During the following X1, data is clocked into the ATC from the PM bus for store operations. The PMWTs are byte pointers assigned as follows:  $\overline{PMWT0}$  - Byte 0,  $\overline{PMWT1}$  - Byte 1,  $\overline{PMWT2}$  - Byte 2,  $\overline{PMWT3}$  - Byte 3.

The ATC has a special mode of operation called Virtual Equals Real. This mode forces the virtual address translation and the translation checking to be inhibited, so that the virtual address is handled as a real 24 bit address.

A clock by clock description of memory operations is given in Chapter 3.

## Real Memory Operations

A Real Memory operation can be initiated by the processor and by an I/O device on the PM Bus. (The ATC can also initiate a real memory operation after successfully translating a virtual address into a real address). The operation is initiated by a device assertion of  $\overline{\text{MAE}}$  (Memory Address Enable) and an address on the PM Bus during X0. The ATC detects the  $\overline{\text{MAE}}$  line and asserts the  $\overline{\text{REQ0}}$  line to secure access to the PM Bus for the next cycle. If a fetch, the ATC clocks the address into its Real Address Register.

The ATC examines the write tags (bits 28-25 of the address message) to identify the type of memory operation (in the same manner as the PMWT0-3 lines are examined during virtual memory operations). The PM Bus bits are assigned as byte pointers as follows:  $\text{PMBUS28}$  - Byte 0,  $\text{PMBUS27}$  - Byte 1,  $\text{PMBUS26}$  - Byte 2,  $\text{PMBUS25}$  - Byte 3.

In order to distinguish between CPC and other device-initiated real memory operations, the ATC hardware examines the  $\overline{\text{PVT}}$  (Processor Virtual Transfer) line during the X1 following  $\overline{\text{MAE}}$  assertion during X0. If  $\overline{\text{PVT}}$  is asserted at the end of the X1, a processor-initiated operation is indicated. The ATC asserts  $\overline{\text{TRAP}}$  if it detects an uncorrectable memory error during a processor-initiated memory fetch.

## Memory Refresh Operation

Each refresh operation consists of a row refresh and a word fetch/check/correct followed by a full store operation. These actions maintain the proper charge levels in the main memory cells, and scrub a word of main memory to correct a single bit error that it may contain.

The ATC performs a refresh to a row of main memory approximately every 16 microseconds, depending on the system clock and timer programming. Thus the entire main memory (256 rows) is refreshed every 4 ms by the memory interface term RAS. In addition to a row refresh, the word fetch/check/correct and store operation scrubs each word bank of main memory every 1.048 seconds (assuming each bank of local memory consists of 64K words).

The Time-Of-Day register/counter is used to determine the 16 microsecond refresh intervals. At the start of the 16 microsecond interval, the ATC tests to see if a refresh operation can be initiated on the PM Bus. If during X0 no PM Bus transfers are initiated (i.e., neither a memory transfer nor an External Register Unit transfer),

and if no double memory error from a previous operation is pending, the ATC asserts  $\overline{REQ0}$  and starts a four cycle refresh operation. During the following X0, the ATC asserts the time-of-day counter value on the PM Bus as the address that is to be refreshed (the refresh bit, PM Bus bit 29 is set and the write tags are reset). The ATC asserts  $\overline{MAE}$ , and asserts  $\overline{REQ0}$  to secure access to the PM Bus the next cycle.

The Memory Interface uses the lower 10 bits of the Time-Of-Day register to activate its RAS term to refresh the Main Memory, and the upper 14 bits to activate the CAS term for reading the word to be scrubbed.

The ATC latches the word to be scrubbed at the end of a subsequent X0, checks it with its Error Correction Check logic, and asserts  $\overline{REQ0}$  to secure access to the PM Bus for the store operation that may follow. If there is no error in the fetched word, the same data is written back into memory. If a single bit error is detected, correction is made, new syndrome bits are generated, and the corrected word is stored.

If a double error is detected, the ATC aborts the operation and returns to its Idle state. The  $\overline{TRAP}$  signal is not asserted, but  $\overline{MDEE}$  is asserted during X1, signaling to the Memory Interface that the write back is to be aborted. In all cases the ATC assumes that the Memory Interface will hold the refresh address it received for the duration required to successfully complete the store.

If a refresh condition is decoded at the same time as either a memory transfer or an External Register Unit transfer is initiated, the refresh operation is postponed. After completing a memory or an External Register Unit transfer, if a memory error trap condition does not exist, the ATC initiates the 4 cycle refresh operation. If the memory error trap condition does exist, the ATC returns to the Idle state.

Note that refresh can be inhibited by any of the following:  $\overline{INH}$  input signal asserted,  $\overline{PMRST}$  input signal asserted, the double bit memory error trap bit set in the Interrupt/Trap Array, and a TOE operation to the Syndrome Register.

## ERROR CHECK/CORRECTION AND SYNDROME BIT GENERATION (ECC)

Associated with each word in memory are 7 syndrome bits. These syndrome bits are generated by the Error Check and Correction (ECC) logic during each memory store operation, and are stored along with that word. During a memory fetch operation, the 7 syndrome bits corresponding with the fetched word are used to check the integrity of the word.

The ECC logic is capable of detecting and correcting single bit errors (including syndrome bit errors), but can only detect and report double and some multiple bit errors.

The ECC logic can be disabled by a bit in the Control Array (CA7=1). The ECC syndrome generation logic can be disabled by use of the Syndrome Register in a diagnostic mode.

### **ECC Generation During Memory Store**

Each bit on the PM Bus is identified by a unique seven bit code. In the syndrome generate mode these Correction Codes are identical to the PMCHK7-1

The ECC logic is normally in the syndrome generate mode and is essentially an odd parity generator. Fourteen data bits associated with each Correction-Code bit are input to an odd parity generator. If the 14 bits by themselves constitute odd parity, the PMCHK line is negated. If these 14 bits constitute even parity, the PMCHK line is asserted. The 7 PMCHK line states obtained in this fashion are then stored in memory along with the data word, and are designated the 7 syndrome bits for that word.

### **Error Check and Correction During Memory Fetch**

The ECC logic goes active only if CA7 in Control Array #2 is not disabling ECC and the operation being performed is a “normal” +0%(i.e., not diagnostic, implying manipulation of the syndrome register) fetch, a fetch for a partial store operation or a fetch for refresh operation.

Whenever a word is fetched from memory, parity is checked for each of the 7 sets of 14 data bits plus the corresponding syndrome bit. If the result of each check is odd parity, all Correction-Code lines remain negated. If a check results in even parity, then the associated Internal Correction-Code line is asserted. The state of the correction-code lines determines whether an ECC error has been detected and the nature of the detected error. All the code lines are negated after an error free fetch. A single Correction-Code line asserted indicates an error in the syndrome bits. Three Correction-Code lines asserted indicate a single bit error, and the incorrect bit can be decoded using ECC logic equations. Any other combination of Correction-Code lines being asserted indicates multiple bit errors.

During fetch operations data is latched during X0 into the ATC. At the same time the ATC ECC logic begins checking the data. If no errors are detected, the data is sent to the requesting device. If either a single data bit or syndrome bit error is detected, the correction is made by the ATC and the corrected data is sent to the requesting

device. With or without errors, the ATC outputs the data that X1 (same cycle). If multiple bit errors are detected, the ATC flags the system by asserting the  $\overline{\text{MDEE}}$  (Memory Data Enable/Error) line at X1, indicating that this is an uncorrectable error condition. If the fetch was initiated by the processor, the ATC sets the Memory Error Trap bit in the Interrupt/Trap Array, which activates the  $\overline{\text{TRAP}}$  line. The trap bit will not be set for fetch, partial store, refresh, or non-processor initiated operations.

After an error has been flagged, the ATC resumes normal flow of state sequencing. It is the responsibility of the Memory Interface logic to monitor  $\overline{\text{MDEE}}$ , and to abort the data write-back if  $\overline{\text{MDEE}}$  is asserted during X1.

In all detected memory error cases (single and multiple bit errors), the ATC  $\overline{\text{MEMERR}}$  line is asserted during X1. This signal is asserted by the ATC as a feature that may be used to count all memory errors for test and evaluation purposes.

### Multi-Word Fetch/Correction

A normal memory fetch, as seen by the ATC, is an address asserted during X0, followed some subsequent X0 by the fetched memory data. An input control signal,  $\overline{\text{DIE}}$ , is used to indicate to the ATC that this is the X0 during which the fetched data is valid on the PM Bus. The ATC then reads the data and corrects it, if necessary, then outputs the data the following X1. If the data has an uncorrectable error, the ATC asserts  $\overline{\text{MDEE}}$  the same X1; and if the fetch was processor initiated, the ATC asserts  $\overline{\text{TRAP}}$ .

For multiple-word fetches the ATC requires the same address and  $\overline{\text{DIE}}$  assertion during subsequent X0 clocks. But the  $\overline{\text{MAE}}$  signal must also be asserted with  $\overline{\text{DIE}}$  to keep the ATC in the fetch/correct state (i.e., state 2) until the last fetch data is read. For the last fetch data read  $\overline{\text{MAE}}$  is not asserted, allowing the ATC to “unlock” from the fetch/correct state.

The ATC asserts  $\overline{\text{MDEE}}$  to indicate an uncorrectable memory error only during the X1 clock of the cycle in which the uncorrectable word is transferred. For CPC-initiated multiple-word fetches, the ATC asserts  $\overline{\text{TRAP}}$  in response to any uncorrectable error.

### EXTERNAL REGISTER UNIT OPERATIONS

The ATC identifies an External Register Unit (ERU) operation by the assertion of  $\overline{\text{ERE}}\overline{\text{P}}$  (External Register Enable/Permit) during X0. The ATC responds by decoding the least significant seven bits on the PM Bus during X0 to determine whether an ATC External Register Unit is being addressed. A transfer to/from an External Register Unit occurs during X1. The direction of transfer is



established at X0 by the state of PM Bus bit 8. If bit 8 is set, an External Register Unit is written from the PM Bus during X1. If bit 8 is clear, the ERU is read during X1.

All External Register Unit transfers are one cycle operations. However, three External Register Unit transfer-out operations to the ATC (External Register Units 52,54,55) can not be immediately followed (for one cycle) by either an External Register Unit operation that requires the ATC, or by a memory operation.

The ATC External Register Units are divided into two groups: External Register Units associated with the virtual operations, and External Register Units for special purpose.

## TIME OF DAY/INTERVAL MONITORING

The ATC monitors the Time-Of-Day via a four microsecond resolution Time-Of-Day register/counter. If a specific time or interval of time is to be monitored, the Interval Timer Monitor Register must be initialized for that interval. When there is an equality between Time-Of-Day register/counter and the Interval Timer Monitor Register, the Time-Of-Day Interrupt bit is set.

## VIRTUAL ADDRESS MONITORING

The ATC features four types of address monitoring using one monitor register and four control bits. Table 5-3 presents a summary of the monitor functions.

When the ATC detects any of the monitored conditions, it aborts the memory operation.

Control Array				Transfer*	AMR Match	Additional Match	Match
8	4	3	2	F4X-AMR	Required	Qualification	Result
0	0	0	1	No	Yes	Virt. Fetch for Execute	BP
0	0	1	0	No	Yes	Virtual Store	MM
0	1	0	0	Yes	No	None	None
0	1	1	0	Yes	Yes	Virtual Store	F4X
1	0	0	0	No	No	Virtual Store	MM

\*Note: Control Array bit 4 enables the last Fetch for Execute Address (F4X) to be transferred to the Address Monitor Register (AMR).

BP — is the Virtual Breakpoint Interrupt (I/TA8)

MM — is the Monitor Match Interrupt (I/TA7)

F4X — is the Fetch for Execute Monitor Interrupt (I/TA6)

GIMTE5005A

Table 5-3 Monitor Functions

### **Breakpoint (Fetch for Execute) Monitor — CA2**

If the Control Array enables monitoring for breakpoint (CA2=1), a Fetch for Execute sets the Virtual Break Point Interrupt if the virtual address matches the value in the Address Monitor Register.

### **Address Monitor (Stores) — CA3**

The ATC has the capability of monitoring all virtual addresses used for full or partial stores. If the Address Monitor Register is set to a specific address and the Control Array enables monitoring (CA3=1), all virtual address for full or partial stores are compared to the address in the Address Monitor Register. If there is a match the Monitor Match Interrupt is set.

### **Fetch for Execute Monitor (Stores)—CA3,4**

If a (virtual) Fetch for Execute is initiated ( $\overline{\text{PMBUS02}}$ ,  $\overline{\text{PMBUS01}} = 1,1$ ) and the Control Array enables monitoring for Fetch for Execute addresses (CA4=1), the virtual address received from the processor at X0 is loaded into the Address Monitor Register. Subsequently, if the Control Array enables monitoring (CA3=1), all virtual addresses relating to full or partial stores are monitored. A Fetch for Execute Monitor Interrupt is set during X1 if there is a match.

### **Trace (Store) Monitor — CA8**

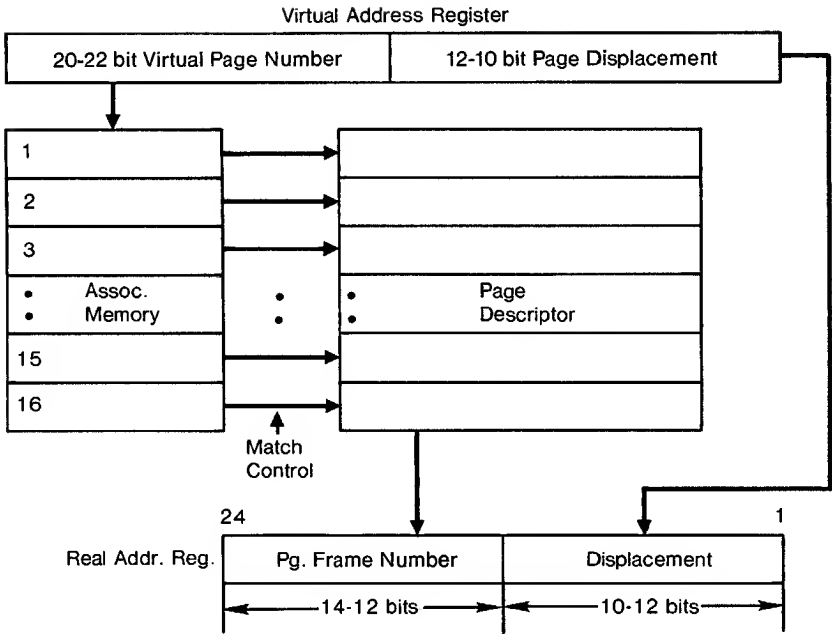
If the Control Array enables monitoring for Virtual Memory Store Trace (CA8=1), then any virtual store triggers a Monitor Interrupt.

## **DYNAMIC ADDRESS TRANSLATION UNIT**

An ATC Dynamic Address Translation Unit model is shown in Figure 5-3.

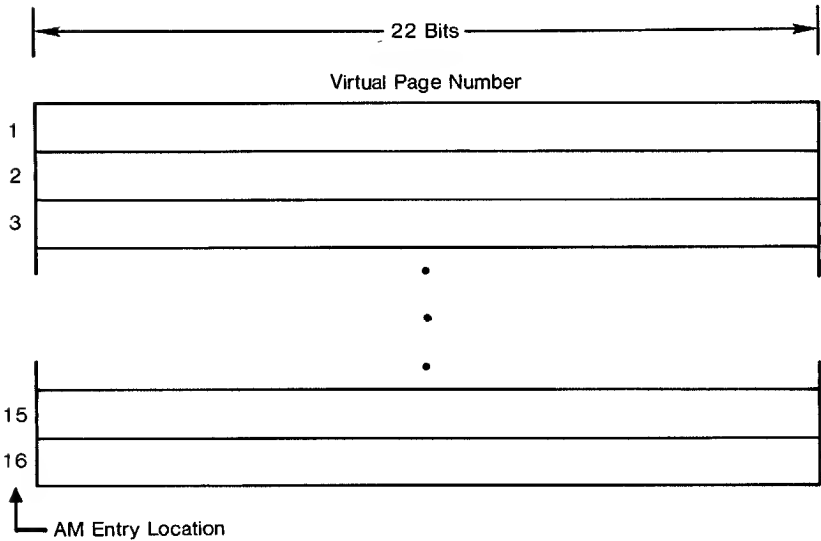
The Dynamic Address Translation (DAT) unit contains 16 entries. Each entry is comprised of two parts: Associative Memory (AM, see Figure 5-4) and corresponding Page Descriptor (see Figure 5-5).

Each AM entry consists of a 22 bit Virtual Page Number. The Page Descriptor is comprised of 25 bits: 8 Protection Check bits, a 14 bit Page Frame Number, 2 AM entry control bits (Changed Page and Register Referenced bits), and an Invalid Register (entry) bit. The DAT is designed to support 3 page sizes (1K, 2K, and 4K bytes). The ATC Virtual Operation flow is shown in Figure 5-6.



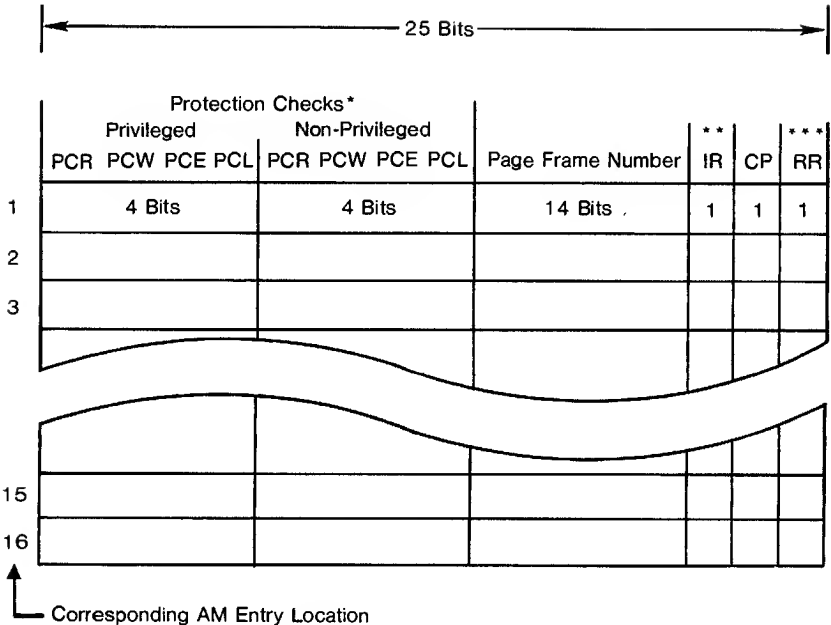
GIM5007B

Figure 5-3 Dynamic Address Translation Unit Model



GIM5008A

Figure 5-4 Associative Memory Virtual Page Numbers



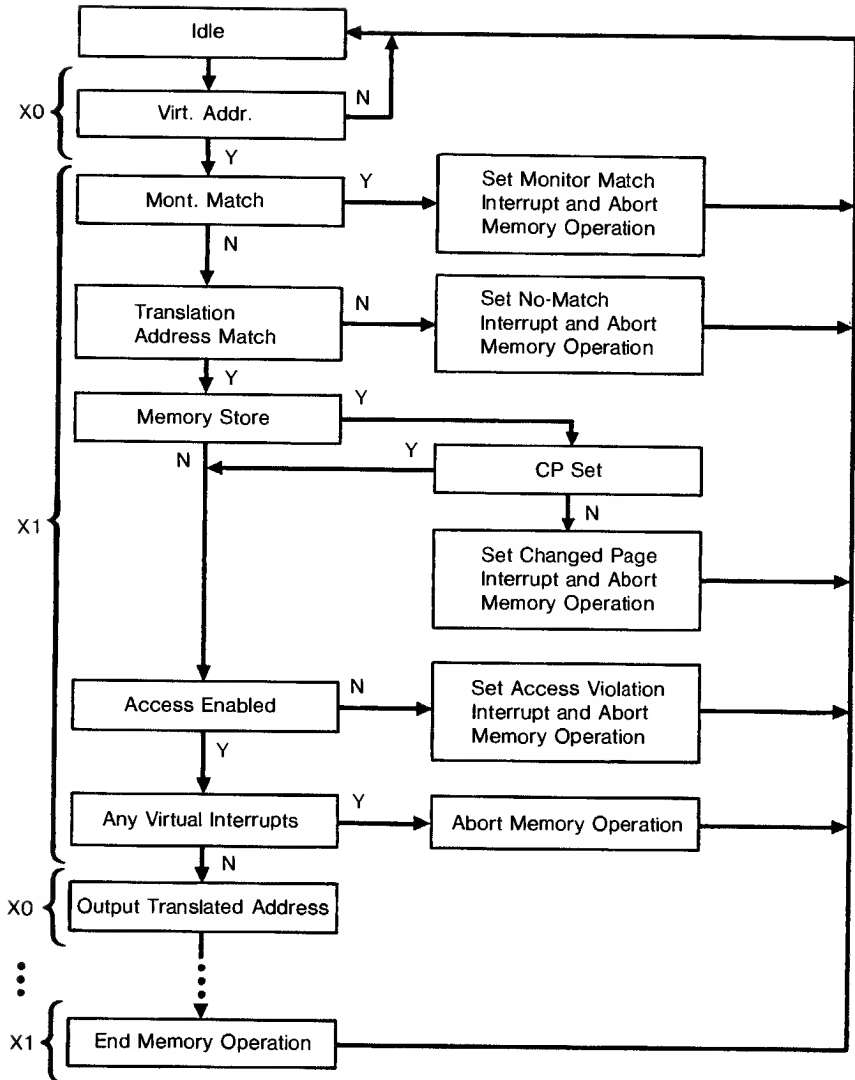
\*PCR, PCW, PCE, PCL are Protection Check bits (defined in a subsequent section on Protection Check).

\*\*The IR bit is accessible for reading.

\*\*\*The RR bit is not accessible for reading or writing.

GIM5009A

Figure 5-5      Page Descriptors



GIM5011

Figure 5-6 ATC Virtual Operation Flow

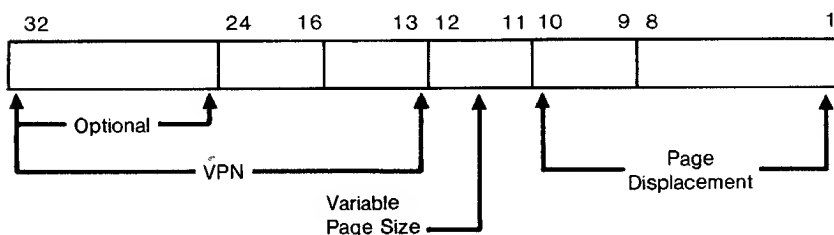
## OVERVIEW

Two types of addresses are presented to the ATC: real, and virtual (addresses requiring translation). When a real memory operation is initiated, the ATC performs no operation on the address (i.e., the real address goes directly to memory). When a virtual memory operation is initiated, the virtual address is latched by the ATC and (if "contained" in the DAT) is translated into a real address. The real address is then asserted on the PM Bus (i.e., passed to memory) by the ATC. If the virtual address was not "contained" in the DAT, the memory operation is aborted and the  $\overline{\text{INT}}$  (Interrupt) line is asserted. This condition is referred to as a DAT fault.

## OPERATION

The ATC can translate either a 24 or a 32 bit (including two protection bits) virtual address into a 24 bit real address.

All virtual addresses are composed of two parts: a Virtual Page Number (called the VPN), and a Page Displacement. As the page size varies from 4K to 1K bytes, the number of bits in the VPN field varies from 20 to 22 bits, and the Page Displacement field varies from 12 to 10 bits. The virtual address format is as follows:



GIM5006

## Translation

When a virtual address is asserted on the PM Bus, the VPN portion is compared with the contents of the AM (i.e., all valid entries, up to 16).

If a match between the virtual address and a valid AM entry is found, the page displacement portion of the virtual address is concatenated with the Page Frame Number from the Page Descriptor of the matched AM entry. The result of the concatenation is a real address. This successful address look-up, and the vector concatenation is called Dynamic Address Translation or simply translation.

The virtual address can be 24 or 32 bits in length. The Address Size bit in the ATC Control Array (CA5) specifies which address length the ATC is to translate. With CA5 set the upper 22 (20) bits of the 32 bits of the virtual address are used by the DAT unit in the associative search. With CA5 reset the upper 8 bits of the virtual address are forced to zeros by the ATC (since these 8 bits are ignored by the hardware, they can be in any state). This prevents a 24 bit virtual address from matching a valid 32 bit address entry in the AM whose upper 8 bits are non-zero.

## Memory Protection

The DAT unit performs four memory protection functions. Each Page Descriptor contains two sets of protection check masks: one for operation in the privileged state, and the other for operation in the non-privileged state. Each set consists of a PCR, a PCW, a PCE, and a PCL bit. The processor state is designated by the Privilege Flag in the Control Array (CA1). If CA1 is set, the privileged set is used. If CA1 is reset, the non-privileged set is used. Each set of protection check masks controls execution of the four access functions; read, write, read for execute, and read for linkage.

An attempt to access memory in a manner violating the selected set of protection checks results in an access violation. If an access violation occurs, the memory operation is aborted and the DAT Access Violation interrupt bit is set in the Interrupt/Trap Array. Table 5-4 indicates which protection is checked for each type of memory operation. A one indicates that the particular protection function is checked, and a zero entry indicates that the protection function is not checked.

Virt. Oper.	Read* Permit (PCR)	Write* Permit (PCW)	Execute* Permit (PCE)	Linkage* Permit (PCL)
Fetch	1	0	0	0
Store	0	1	0	0
Fetch for Execute	0	0	1	0
Fetch for Linkage	0	0	0	1
TOE to ERU 55	**	**	**	**

\*Operation is enabled when the bit is a 1.

\*\*The TOE operation on ERU 55 (called the TVA command) is described in the External Register section dealing with virtual operations (and commands).

GIMTE5010

Table 5-4 Protection Functions for Memory Operations

### **Invalid Register (IR)**

Associated with each VPN is an IR bit. The IR bit is used to indicate whether a particular AM entry contains a valid/invalid entry. If an IR bit is set that entry is invalid and is not used in the translation (association) process.

The IR bit is program accessible for reading.

### **Changed Page (CP)**

The Changed Page (CP) bit indicates that the page referenced by an AM entry has/has not been written into by a previous memory operation. When a memory operation writes into a page that has its CP bit reset, the Change Page interrupt bit is set in the Interrupt/Trap Array register, the memory operation is aborted, and the CP bit for that page is set. A write into a page that already has its CP bit set is treated like any other memory operation: the CP interrupt is not set, and the operation is continued.

The CP bit is program accessible.

### **Register Referenced (RR)**

The Register Referenced (called the RR) bit is used to implement a Least Recently Used algorithm for replacing entries in the DAT. Two situations arise regarding the loading of the AM entries. The first is the loading of the "table" prior to use. The second is the updating of the table when process translation fails. Since the AM has a finite number (16) of entries, system operation will require replacing some unused or old entries with new entries. The RR bits are used to this end. There is one RR bit for each AM entry.

Any AM register that does not have its RR bit set may have a new VPN and corresponding Page Descriptor loaded into that entry location. Once the RR bit is set for an entry, that entry cannot have new data loaded to either the VPN or Page Descriptor. When all RR bits are set, the ATC automatically resets them.

The RR bit is set for an individual AM entry during a successful translation.

The Least Recently Used algorithm implemented in hardware uses a priority scheme. The lowest numbered AM entry (1) has the highest priority, and highest numbered AM entry (16) has the lowest priority. Whenever an entry is added (i.e., loaded) to the AM, the highest priority entry (lowest numbered) whose RR bit is reset is accessed.

Several External Register Unit operations can be used to reset/set RR bits.



The RR bit is not program accessible.

## Protection

Each AM entry has two sets of corresponding memory Protection Check bits. The Protection bits are contained in the Page Descriptor. The Privilege Flag in the Control Array (CA1) selects which set of protection check bits is to be used. Below is a description of each of the memory protection check bits:

1. PCR = Read Permit Bit  
 PCR = 0; read access not permitted  
 PCR = 1; read access permitted  
 The Read Permit Bit determines whether a read access is permitted into a page.
2. PCW = Write Permit Bit  
 PCW = 0; write access not permitted  
 PCW = 1; write access permitted  
 The Write Permit Bit determines whether a write access is permitted into a page.
3. PCE = Read for Execute Permit Bit  
 PCE = 0; read for execute access not permitted  
 PCE = 1; read for execute access permitted  
 The Execute Permit Bit determines whether a read for execute is permitted into a page.
4. PCL = Read for Linkage Permit Bit  
 PCL = 0; read for linkage access not permitted  
 PCL = 1; read for linkage access permitted  
 The Linkage Permit Bit determines whether a read for linkage is permitted into a page.

The Protection Check bits are program accessible.

## Page Frame Number

The Page Frame Number (PFN) portion of the Page Descriptor is 14 bits wide. This accommodates the 1K page size, which requires the largest number of PFN bits. When a virtual address presented to the DAT has an AM match with no protection check violation, the PFN is concatenated with the page displacement portion of the virtual address to form the real address. The number of PFN bits actually used to form a real address is dependent on the virtual page size (1K page - 14 bits, 2K page - 13 bits, 4K page - 12 bits).

The Page Frame Number is program accessible.

## **Virtual Page Number**

The VPN (Virtual Page Number) is the 22 bit content of each of the 16 AM entries used in the Dynamic Address Translation (DAT) Unit. An AM search consists of simultaneously comparing the VPN portion of a virtual address against all of the valid VPNs in the AM. A successful search means that one of the AM entries matches (i.e., is equal to and is valid) the virtual address.

The Virtual Page Number can be loaded, but not read.

## **Interrupts**

Four conditions that can abort the conversion of the virtual address to a real address before completion are: a No Match, an Access Violation, a Changed Page, and an Address Monitor match.

The No Match condition occurs when the VPN of a virtual address does not match any of the VPNs in the AM. The Access Violation condition occurs when there is a match, but the memory operation violates the appropriate protection check. A memory operation that results in a No-Match sets the DAT No-Match Interrupt bit which aborts the memory operation. An Access Violation sets the DAT Access Violation Interrupt bit which aborts the memory operation.

A Changed Page condition occurs the first time a page is written into, at which time the Changed Page Interrupt bit is set and the memory operation is aborted.

The Address Monitor match occurs when the virtual address meets the conditions for an Address Monitor match, forcing a memory operation abort (see Address Monitor Register description).

A memory operation is aborted by no assertion of  $\overline{\text{MAE}}$  in the cycle following the virtual address.

In all of these cases the virtual address causing the exception condition is placed in the Virtual Address Register Buffer.

When the Virtual Equals Real bit is set in the Control Array the DAT No Match, the DAT Access Violation and the Changed Page interrupts are inhibited. Only interrupts related to address monitoring can be asserted.

## **EXTERNAL REGISTER DEFINITIONS**

The ATC has two groups of External Register Units (ERUs): ERUs for special purpose, and ERUs associated with virtual operations. Associated with some ATC virtual operations are associative memory commands. See Table 5-5 for the ATC ERU assignments.

For a detailed description of ERU transfer operation, see Chapter 3 of this manual. The numerical decode for each ERU will first be given in decimal, followed in parenthesis by its hexadecimal equivalent with a capital "H" appended (e.g., decimal twenty: 20 (14 H)). When an ATC ERU is to be read, the term TIE will be used (Transfer-In-External). When an ATC ERU is to be written, the term TOE will be used (Transfer-Out-External). All TIE and TOE operations are one cycle, unless otherwise specified. All ERUs are addressed with only the lower seven address bits. The eighth bit determines whether a TIE or a TOE operation is to be executed.

*7 Bit ERU # (Dec.)      (Hex.)		Transfer Out* Operation (Bit 8 = 1)	Transfer In* Operation (Bit 8 = 0)
40	28	Control Array #2	Control Array #2
41	29	I/T Array	I/T Array
42	2A	Interrupt Mask Reg	Interrupt Mask Reg
43	2B	Intvl. Timer/Monitor Reg	Intvl. Timer/Monitor Reg
44	2C	Time-Of-Day Reg/Counter	Time-Of-Day Reg/Counter
45	2D	Address Monitor Reg	Address Monitor Reg
46	2E	Bus Interrupt Reg	Bus Interrupt Reg
47	2F	Write Page Size (WPS)	Read Page Size (RPS)
48	30	Invltd. Assoc Mem (IAM)	Enabl & Set Pg Frm (ESPF)
49	31	Write Syndrome Bits	Read Syndrome Bits
50	32	Write Virtual Page (WVP)	Read Page Frame (RPF)
51	33	Write & Set Pg Frm (WSPF)	Clear Assoc Mem (CAM)
52	34	Purge Selective (PS)	Not Used
53	35	Write Purge Mask (WPM)	Read Purge Mask (RPM)
54	36	Purge Set W Mask (PSM)	Read Virtual Address (RVA)
55	37	Trans Virt Addr (TVA)	Read Real Address (RRA)

\*Only a seven bit address is shown, eight bits are required. The direction of transfer (in/out) specifies the eighth bit. Therefore, all transfer out addresses are different from those directly indicated by table (e.g., 40 (28 H) —(A8 H) for the transfer-out operation, the transfer-in is correct as shown).

GIMTE5012A

Table 5-5      ATC ERU Assignments

TIE and TOE are CPC op code mnemonics of instructions which are executed to read (TIE) and write (TOE) registers external to the CPC. The reader should refer to Chapter VI of this manual for further information on these op codes.

## SPECIAL PURPOSE ERU

The ATC special purpose ERU is used for ATC control.

### Control Array #2

Control Array #2 is a 16 bit register used to select modes of operation of the ATC. It can be accessed with a TIE or TOE operation on

ERU 40 (28 H). Control Array #1 is located in the CPC. (See Chapter 4 for details.)

The Privilege Flag can be set or reset with a TOE on ERU 40 and with an external ATC pin (PRIV). Activating PRIV during X0 enables setting/resetting the Privilege Flag in the subsequent X1. Following the enable, if during X1 PRIV is activated, the Privilege Flag is set; if during X1 PRIV is not activated, the Privilege Flag is reset. The contents of the Control Array are shown in Table 5-6.

16	11	10	9	8	7	6	5	4	3	2	1
NU	Res	NM	VER	STr	ECC	PCK	ASz	F4X	Mon	BP	Prv

1. Privilege Flag
2. Break Point Enable
3. Monitor Enable
4. Fetch for Execute Monitor Enable
5. Address Size 24/32
6. No Protection Check
7. ECC Disable
8. Virtual Memory Store Trace Enable
9. Virtual Equals Real
10. No Match Disable
11. Reserved For Future Use
12. Reserved For Future Use
13. Not Used
14. Not Used
15. Not Used
16. Not Used

GIMTE5013A

Table 5-6 Control Array Representation

**Control Array Definition** — In subsequent descriptions, the bits in Control Array #2 are identified by “CA” followed by the bit number (e.g., Control Array #2 bit 5 = CA5).

Note that there is only one monitor register and several modes of operation for monitoring. This results in some mutually exclusive conditions. These are noted below.

CA1 : Privilege Flag

This indicator shows which set of protection bits in the Page Descriptors is used. If this indicator is set the ATC uses the privileged set, and if this indicator is reset the ATC uses the non-privileged set.

- CA2: Break Point Enable**  
This indicator enables/disables the Virtual Breakpoint Interrupt logic in the ATC. If this bit is set an interrupt is asserted when the current virtual address specifies Fetch for Execute (PMB02,01=1,1). Note that CA2 and CA4 are mutually exclusive.
- CA3: Monitor Enable**  
This indicator enables/disables the ATC hardware compare of the virtual addresses used during writes to memory with the address in the Address Monitor Register. If this bit is set, an interrupt is set when the current virtual address matches the contents of the Address Monitor Register. Note that CA3 and CA8 are mutually exclusive.
- CA4: Fetch for Execute Monitor Enable**  
This indicator, when set, forces the ATC hardware to retain the last address used to fetch from memory with Fetch for Execute specified (by the two least significant bits of the virtual address). Consequently, that address is loaded into the Address Monitor Register. When CA3 is set, this address is monitored for all writes to memory. Note that CA4 and CA2 are mutually exclusive.
- CA5: Address Size 24/32**  
When this indicator is reset, all virtual addresses are handled as 24 bits, and byte 0 is forced clear. When this indicator is set, all virtual addresses are handled as 32 bits.
- CA6: No Protection Check**  
When set, this indicator inhibits protection bit checking in the DAT during a virtual memory operation.
- CA7: ECC Disable**  
When set, this indicator inhibits the Error Check/Correction circuitry in the ATC, and inhibits setting of bit 17 of the Syndrome Register (memory error indicator).
- CA8: Virtual Memory Store Trace Enable**  
When this bit is set and Monitor Enable is reset (CA3=0), any virtual memory store is aborted. Note that CA3 and CA8 are mutually exclusive.
- CA9: Virtual Equals Real**  
When set, all memory addresses (virtual and real) are handled as real addresses. Virtual Memory messages require the same number of cycles regardless of this bit.

CA10: No Match Disable  
When set, the DAT No Match interrupt is disabled.

Interrupt/Trap Array

The Interrupt/Trap Array (I/T Array), shown in Table 5-7, is a 12 bit register that indicates the ATC Interrupt and Trap conditions. This register can be accessed with a TIE or TOE operation on ERU 41 (29 H).

16		12				8				4					
PU	0	ME	PT	SP	AV	NM	CP	BP	MM	F4X	BIN	TOD	0	0	0

- 1. Not Used
- 2. Not Used
- 3. Not Used
- 4. Time-Of-Day Interrupt
- 5. P-M Bus Interrupt (BIN)
- 6. Fetch for Execute Monitor Interrupt
- 7. Monitor Match Interrupt
- 8. Virtual Break Point Interrupt
- 9. Changed Page Interrupt
- 10. DAT No-Match Interrupt
- 11. DAT Access Violation Interrupt
- 12. Special Interrupt
- 13. Programmable Trap\*
- 14. Memory Error Trap\*
- 15. Not Used
- 16. Power-Up/Reset Trap\*

\*Trap conditions cannot be masked, but all interrupts can be masked by setting the appropriate bits in the Interrupt Mask Register.

GIMTE5014A

Table 5-7 Interrupt/Trap Array Representation

**Trap and Interrupt Operation** — The ATC can source an Interrupt line ( $\overline{INT}$ ) and a Trap line ( $\overline{TRAP}$ ). When there are one or more interrupt conditions met in the ATC as indicated by the I/T Array flags, the ATC asserts  $\overline{INT}$  (Interrupt) after X1, signaling the processor that an interrupt requires servicing. The processor responds with a TIE operation (read) to the I/T Array, and by servicing the interrupt that has the highest priority. Upon completion, the processor executes a TOE operation (write) to the I/T Array with the identical state that was read in response to the interrupt, but with the specific interrupt (bit) that was serviced cleared. At this point, if no other interrupts are pending, the ATC deactivates the  $\overline{INT}$  line. Any interrupt can be masked by setting the corresponding bit in the Interrupt Mask Register (however, if any interrupt that would normally result in the memory operation being aborted is masked, the memory operation takes place).

The interrupts are divided into two categories. The first consists of the "Virtual Interrupts" corresponding to bits 6 through 11 of the I/T Array. When these are asserted, the ATC aborts the virtual memory operation that was initiated, and asserts the  $\overline{\text{INT}}$  line (i.e., the processor is interrupted). The remaining interrupts are in the second category corresponding to bits 4, 5, and 12 of the I/T Array. These interrupts occur randomly to processor operation and are serviced as they occur. The ATC does not interrupt the processor during a processor initiated fetch operation until the fetched data is received by the processor (or for a virtual fetch that faults until the second cycle after the virtual address is asserted).

If the I/T Array is read by any device, any subsequent PM Bus (BIN) interrupt is not asserted until a TOE operation to the I/T array is completed. Upon completion of the TOE operation, the pending interrupt is recognized. Other interrupts are not "buffered" in this way; therefore, the programmer must be aware of their potential recognition by the ATC between a TIE and a TOE operation to the I/T Array, so as not to inadvertently clear an unserviced interrupt via a TOE to the I/T Array.

Traps cannot be masked in the ATC. The ATC will activate the  $\overline{\text{TRAP}}$  line after X1 in response to: the Power-Up/Reset sequence; an uncorrectable memory error during a processor initiated fetch; a TOE operation that sets the Programmable Trap bit in the I/T Array. The  $\overline{\text{TRAP}}$  line can be activated independently of the processor fetch operation (real or virtual).

**Trap and Interrupt Definition** — In the following descriptions the bits in the I/T Array are identified by "I/TA" followed by the bit number (e.g., I/T Array bit 12: I/TA 12). The following descriptions are presented in the context of the interrupts being enabled (i.e., not masked).

#### I/TA 4: Time-of-Day Interrupt

This interrupt bit is set at X1 when the Time-Of-Day counter value matches the contents of the Interval Timer/Monitor Register. The TOD Interrupt remains asserted until a TOE operation to the I/T Array clears this bit and either 4 microseconds has elapsed or the Interval Timer/Monitor Register no longer matches the Time-of-Day counter value. Since a TOD Interrupt condition is asynchronous to CPC operation, the ATC user should be aware that a TOD Interrupt bit can set in the I/T Array while the Processor is already in an interrupt service routine. Consequently the time between a TIE

operation and a TOE operation to the I/T Array in response to an interrupt must be less than 4 microseconds to guarantee not losing a TOD interrupt.

TOD Interrupt recognition is inhibited during fetch operations, and during X1 of both TOD and ITMR writes.

**I/TA 5: PM Bus Interrupt**

This interrupt bit is set during X1 when a device on the PM Bus sends a message to the processor Bus Interrupt (BIN) register. The loading of the BIN register sets this bit. If, however, at the time that the BIN register is loaded an interrupt is currently being serviced (a TIE operation on the I/T Array and the BIN register had been performed), then this bit will not be set in the I/T Array until the asserted interrupt bit is reset (a TOE operation on the I/T Array is performed with I/TA 5 reset).

The routine that services this interrupt must read the BIN register before attempting to write it.

**I/TA 6: Fetch for Execute Monitor Interrupt**

This interrupt bit is set if the Fetch for Execute Monitor Enable bit (CA4) and the Monitor Enable bit (CA3) are both set, and the address in the Address Monitor Register (last Fetch for Execute) matches the virtual address during a virtual write. When this bit sets, the memory operation is aborted.

This interrupt is also set when the Virtual Equals Real bit is set (CA9=1) if the above conditions are met.

**I/TA 7: Monitor Match Interrupt**

This bit is set during a virtual write to memory if the virtual address matches the address in the Address Monitor Register, the Fetch for Execute Monitor Enable bit (CA4) is reset, and the Monitor Enable bit (CA3) is set. The Address Size bit (CA5) is used to determine if the match is to be for 24 or 32 bit virtual addresses.

The Monitor Match Interrupt bit is also set during any virtual write to memory if the Virtual Memory Store Trace bit (CA8) is set, and the Monitor Enable bit (CA3) is reset. The Address Monitor Register contents need not match the virtual address to set the interrupt.

This interrupt bit is also set when the Virtual Equals Real bit is set (CA9=1) if the above conditions are met.



When the Monitor Match Interrupt bit is set, the memory operation is aborted.

I/TA 8: Virtual Break Point Interrupt

This interrupt bit is set during a virtual memory Fetch for Execute operation if the Breakpoint Enable bit (CA2) is set, and the virtual fetch address matches the address in the Address Monitor Register. The memory operation is aborted if this bit sets.

This interrupt is also set when the Virtual Equals Real bit is set (CA9=1) if the above conditions are met.

I/TA 9: Changed Page Interrupt

This interrupt bit is set when a page is written into for the first time. The memory store that sets this bit is aborted.

I/TA 10: DAT No-Match Interrupt

This interrupt is set when the virtual address does not match any of the entries in the associative memory. When this bit sets, the memory operation is aborted.

If the No Match Disable bit is set (i.e., CA10), the interrupt bit is allowed to set but the interrupt signal (i.e.,  $\overline{\text{INT}}$ ) is not asserted.

I/TA 11: DAT Access Violation Interrupt

This interrupt bit is set when there is an access violation as determined by a message comparison with the Protection Check bits (i.e., the address association was successful, but the operation is not permitted). When this bit sets, the memory operation is aborted.

I/TA 12: Special Interrupt

This bit is set when  $\overline{\text{SPINT}}$  is asserted (see Table 5-1, ATC pin description).

I/TA 13: Programmable Trap

The Programmable Trap is set with a TOE operation to the I/T Array. The setting of this bit forces the  $\overline{\text{TRAP}}$  line to activate immediately.

I/TA 14: Memory Error Trap

When a processor initiated memory fetch (all virtual and some real fetches are processor initiated) has an

uncorrectable memory error, the Memory Error Trap bit is set, forcing the  $\overline{\text{TRAP}}$  line to be activated. Refresh is disabled while this bit is set.

**I/TA 16: Power-Up Reset Trap**

The Power-Up/Reset Trap bit is set by the assertion of the  $\overline{\text{PMRST}}$  signal (normally, as part of the power-up sequence). When this bit is set, the  $\overline{\text{TRAP}}$  line is activated.

**Interrupt Mask Register (IMR)**

The Interrupt Mask Register (IMR) allows the masking of all interrupt bits in the I/T Array (i.e., bits 12-4). Note that the trap bits can not be masked. If a bit is set in the IMR then the corresponding bit in the I/T Array cannot be set, and the  $\overline{\text{INT}}$  line is not activated for that interrupt condition. Consequently, when a mask bit is set, any interrupt condition that would otherwise result in a memory operation being aborted proceeds normally.

The IMR is program accessible with TOE and TIE operations on ERU 42 (2A H).

**Interval Timer/Monitor Register (ITMR)**

The Interval Timer/Monitor Register (ITMR) is a 32 bit register that is loaded with a desired "interval-of-time" value. Logic compares the content of the ITMR with the contents of the Time-Of-Day register. When there is a match between the ITMR and the Time-Of-Day register, the Time-Of-Day interrupt is set (I/TA 4=1). If the Time-Of-Day interrupt bit sets during a memory fetch operation,  $\overline{\text{INT}}$  is not asserted until the fetch operation has been completed. ITMR can be accessed with a TIE or TOE operation on ERU 43 (28 H).

**Time-Of-Day Register/Counter (TOD)**

The Time-Of-Day register/counter (TOD) is a 32 bit counting register with a 4 microsecond clocking (count) rate. It is driven by a 250 KHz external oscillator. The TOD can be accessed with a TIE or TOE operation on ERU 44 (20 H).

Note that a TOE operation to the TOD must not change the contents of bits 10 thru 1 of the TOD if these bits are used to refresh memory. Consequently a TIE operation from the TOD should precede the TOE operation, and bits 10-1 extracted and concatenated to the new value before the TOE operation is executed.

### Address Monitor Register (AMR)

The Address Monitor Register (AMR) is a 32 bit double stage register, used to monitor virtual addresses during virtual (or Virtual Equals Real) memory operations. The AMR can be loaded two ways: either directly with a TOE operation, or indirectly with a virtual Fetch for Execute operation with the Fetch for Execute Monitor Enable bit set (CA4=1, Fetch for Execute is indicated by  $\overline{\text{PMBUS02}}, \text{PMBUS01} = 1,1$ ). The virtual address is loaded into the AMR. In the 24 bit mode, byte 0 of the AMR is not compared (therefore, byte 0 is always equal).

The AMR can be accessed with a TIE or TOE operation on ERU 45 (20 H).

### Bus Interrupt Register (BIN)

The Bus Interrupt Register (BIN) is a 32 bit register that is loaded by a device on the PM Bus with a TOE operation. The ATC determines that the message on the bus is for the BIN and, if the  $\overline{\text{EREP}}$  was not asserted the previous X1, latches in the message and asserts  $\overline{\text{EREP}}$  during subsequent X1 clocks. The PM Bus Interrupt bit ( $\text{I/TA5}=1$ ) is set and activates the  $\overline{\text{INT}}$  line after X1. As long as  $\overline{\text{EREP}}$  remains asserted during X1 clocks, the BIN does not latch in messages.

A TIE operation to the BIN forces the  $\overline{\text{EREP}}$  to become negated that X1, allowing writes to the BIN. The BIN is accessible with TIE and TOE operations on ERU 46 (2E H).

The System Interface Controller (SIC) uses the BIN Register to report status to the CPC. When the SIC loads the BIN Register, the ATC interrupts the CPC.

### Syndrome Register (SR)

Access to the Syndrome Register (SR) is strictly for diagnostics. The SR is a 7 bit register used to latch the seven syndrome bits associated with a data word fetched from memory or stored into memory. During a word fetch from memory, the syndrome bits asserted by memory via the PMCHK lines are loaded into the SR.

The SR may be read with a TIE operation. Bits 31 through 25 are the syndrome bits, corresponding respectively to PMCHK7 thru PMCHK1. All other bits are cleared with the possible exception of bit 17. Bit 17 is set if any type of memory error was detected during the last memory access (read); otherwise it is also cleared.

Loading the SR with a TOE Operation disables further loading of the SR with store operations until a TIE operation to the SR is executed (fetch operations continue to load the SR). Memory stores (full or partial) after a TOE operation to the SR do not generate new

syndrome bits. Instead, the ATC asserts on the PM Bus the SR contents resulting from the more recent of the SR TOE operation or the last fetch operation.

When executing TOE stores to the SR, the SR should be released long enough every 16 microseconds to allow the memory refresh operation, and no other subsystems should be allowed to access (write) memory while syndrome bit generation is inhibited (i.e., to prevent extraneous memory errors).

The SR is accessible with TIE and TOE operations on ERU 49 (31 H).

The SR format is:

32	31	25	24	18	17	16	9	8
X	PMCK7-1	X	----	X	Er	X	----	X

X — Don't Care

Er — 1 = Memory Error; 0 = No Memory Error

GIM5015

## Memory Data/Processor Data Register (MD/PD)

The Memory Data/Processor Data (MD/PD) register is not accessible with TIE or TOE operations. This is a double stage, 32 bit holding register for data transferred from/to memory.

The memory data stage of the MD/PD register is used to hold data read from memory during fetch and partial store operations for assertion on the PM Bus the X1 following the data reception.

The processor stage of the MD/PD register latches in CPC data from the PM Bus during X1 of full and partial store operations and, based on memory speed (i.e.,  $\overline{DIE}$  assertion), transfers the data out to memory during a subsequent X1.

## VIRTUAL OPERATION ERUs

The following ERUs are directly associated with virtual address translation.

## Virtual Address Register (VAR)

The Virtual Address Register is a 32 bit double stage register used to latch virtual addresses asserted by the CPC. The first stage is loaded every X0 except the X0 following a TOE operation to ERU 52, 54 or 55 (34, 36 or 37 H). The first stage is also loaded during X1 of TOE Operations to ERU 50, 51, 52, 54 or 55, later described in this chapter as the WVP, WSPF, PS, PSM, and TVA commands. The first stage of the Virtual Address Register is called the VAR. The second stage of the Virtual Address Register is called the Virtual Address Register Buffer (VARB).

The first stage is the “action” stage and the second stage merely holds the address for possible later examination. Addresses are loaded into the second stage (VARB) two ways:

1. During a virtual memory operation, the virtual address is loaded into the VAR, regardless of the state of the Virtual Equals Real bit (CA9), during X0. The VAR contents are loaded into the VARB during the subsequent X1.
2. During X1 of a TOE operation on ERU 55 (37 H) (called the TVA command), the data portion of the transfer is loaded into the VAR. The data is transferred from the VAR into the VARB during the following X1 clock.

Hence the content of the VARB is the last address for which translation was attempted. VARB can be read with a TIE operation on ERU 54 (36 H) (called the RVA command). The least significant two bits of the VARB are not part of the virtual address, but indicate the type of protection that was associated with that particular address. For the format, see Associative Memory Commands—RVA.

### Real Address Register (RAR)

The Real Address Register (RAR) is a 32 bit register. The RAR format is:

32	31	30	29	28	25	24	3	2	1
SP	Not Used		R	WT		Real Address			PC

- SP — Used as Scratch Pad Indicator (set indicates Scratch Pad access)  
 R — Used as Refresh Operation Indicator (set indicates refresh operation)  
 WT — Write Tags — All clear Fetch, All set Full Store, etc.  
 PC — Protection Check Bits for Virtual Memory Operations

GIM5016A

Real addresses are loaded into the RAR during X1 the following ways:

#### 1. Virtual Memory Operation:

Bits 32-29 are not used, except during V.E.R. operation, when they are written with corresponding bits from the virtual address. Bits 24 through 13 are always transferred from the Associate Memory (AM).

## Address Translation Chip (ATC)

Bits 12 and 11 are transferred from the AM or VAR, depending on the page size:

Bit No	Page Size		
	4K	2K	1K
12	VAR	VAR	AM
11	VAR	AM	AM

GIM5017

Bits 10 thru 1 are always transferred from the VAR. The write tags (bits 28 thru 25) are read from the PMWT0-3 lines.

### 2. Real Memory Operation:

All bits are loaded from the PM Bus via the VAR register during fetch operations (store operation addresses are not loaded); bits 28 through 25 are the write tags.

If the Virtual Equals Real bit is set (CA9=1) a virtual address is loaded into RAR as if it were a real address for both fetches and stores. However, the PMWT0-3 lines are still used as the write tags.

3. A TOE operation on ERU 50 (32 H) and a TIE operation on ERU 48 (30H), called respectively the WVP and ESPF commands, force the highest priority entry with its RR bit reset to load its corresponding Page Descriptor (i.e., the Page Frame Number, the Protection Check bits, the Changed Page bit and the Invalid Register bit) into the RAR, and then into the Descriptor Data Register.
4. The TOE Operation on ERU 55 (37 H), called the TVA command, loads the RAR with the Page Frame Number from the Page Descriptor and the page displacement from the VAR, whether the translation is successful or not.

## Descriptor Data Register (DDR)

The Descriptor Data Register (DDR) is a 32 bit register used to hold the contents of the RAR immediately (next cycle) after any virtual memory address operation, TOE operations to either ERUs 50 or 55, and after a memory or TIE operation on ERU 48 (i.e., the WVP, TVA, or ESPF commands, respectively, described in detail in the Associative Memory Commands section).

The DDR acts as a history register for what has occurred in the RAR, related to virtual operations. The DDR retains a copy of the last virtual operation RAR contents. This enables the processor to surrender use of the ATC without losing the RAR contents. The DDR can be accessed with a TIE operation on ERU 50 (32 H), called

the RPF command. The format of the DDR is shown in the Associative Memory Commands section in conjunction with the command utilizing the DDR.

### Associative Memory and Page Descriptor Registers

The AM (Associative Memory) consists of sixteen 22-bit registers, each containing a Virtual Page Number (VPN). Associated with each AM register is a 25-bit Page Descriptor Register with a 12-14 bit (depending on page size) field containing a Page Frame Number (PFN). During translation, the virtual address in the VAR is compared to the AM entries. If there is a match (i.e., direct equality) to a particular VPN, the corresponding PFN bits are asserted on the internal ATC bus and are concatenated in the RAR to the 10-12 lower order bits (depending on the page size) of the VAR.

The VPNs are accessible with a TOE operation on ERU 50 (32 H), called the WVP command. The VPNs are not TIE accessible. A TOE on ERU51, called the WSPF command, accesses the PFN for loading. A PFN can be transferred to the DDR in four ways: a successful virtual translation, a WVP command, a TIE on ERU 48 (called the ESPF command), and a TOE on ERU55 (called the TVA command). A subsequent TIE on ERU 50 (32 H) called the RPF command outputs the PFN to the PM Bus.

Refer to the Dynamic Address Translation section for a detailed description of the translation operation using the VPNs and PFNs.

## ASSOCIATIVE MEMORY COMMANDS

The DAT is controlled with the use of Associative Memory commands. Some of these commands access a register, and some simply trigger a hardware operation. A list of these commands is given in Table 5-8.

Addr *	Command	TIE	TOE
2F	WPS		X
2F	RPS	X	
30	IAM		X
30	ESPF	X	
32	WVP		X
32	RPF	X	
33	WSPF		X
33	CAM	X	
34	PS		X
35	WPM		X
35	RPM	X	
36	PSM		X
36	RVA	X	
37	TVA		X
37	RRA	X	

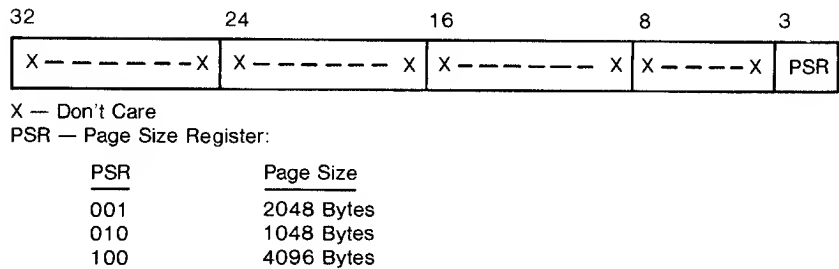
\* 7 Bit hex address

GIMTE5018

Table 5-8 Associative Memory Commands

**WRITE PAGE SIZE (WPS)**

The Write Page Size (WPS) command is a TOE Operation on ERU 47 (2F H). The lower three bits of the transferred data are loaded into the Page Size Register (PSR). Bit 1, 2, or 3 set (exclusive) indicates 2K byte, 1K byte, or 4K byte page size respectively. The WPS command format is shown in Figure 5-7.



GIM5019

Figure 5-7      WPS Format

**READ PAGE SIZE (RPS)**

The Read Page Size (RPS) command is a TIE operation on ERU47 (2F H). For format description, see WPS.

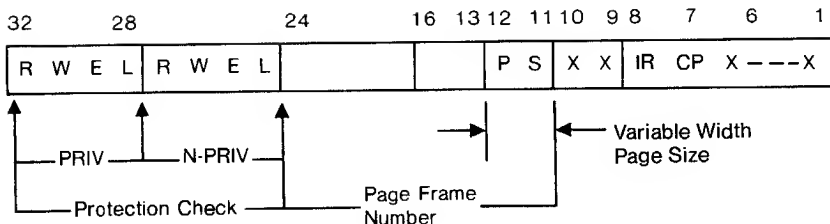
**INVALIDATE ASSOCIATIVE MEMORY (IAM)**

The Invalidate Associative Memory (IAM) command is not a specific register operation. A TOE operation on ERU 48 (30 H) acts as a command to the ATC DAT hardware (i.e., the data portion of the transfer is ignored). The ATC, upon execution of an IAM, sets all IR bits and resets all RR bits in the AM. This has the effect of invalidating and clearing all entries in the DAT.

**ENABLE AND SET PAGE FRAME (ESPF)**

The Enable and Set Page Frame (ESPF) command results in a register to register transfer inside the ATC. A TIE operation on ERU 48 (30 H), called ESPF, acts as a command to the ATC hardware. The data transfer portion of the ESPF is not specified (i.e., is ignored). The content of the Least Recently Used Page Descriptor entry (i.e., the highest priority entry with the RR bit reset) is transferred from that Page Descriptor to the RAR. Then the DDR is loaded from the RAR, and the RR bit for that entry is set. The format of the DDR content is shown in Figure 5-8.





R—Read, Protection Check

W—Write, Protection Check

E—Execute, Protection Check

L—Linkage, Protection Check

PS—The PS (page size) field is the least significant two bits contained in the PFN. Whether both b12 and b11, b12 only, or neither bit is used to form the real address is a function of the page size specified.

X—Don't Care

IR—Invalid Register Bit

CP—Changed Page Bit

GIM5020A

Figure 5-8 DDR Content Format

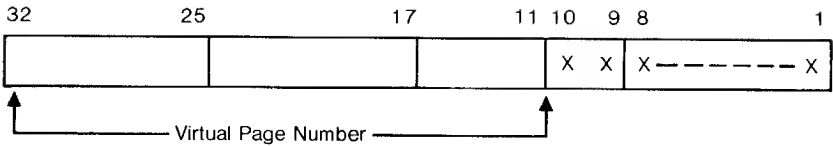
This command takes  $1\frac{1}{2}$  cycles to complete, but is effectively a one cycle operation without restrictions.

The DDR contents resulting from this operation remain unaltered until a virtual address translation (attempt) occurs, or any of 3 commands occurs: TVA, WVP or ESPF.

**Restriction**—The ESPF that accesses the sixteenth entry cannot be immediately followed by another ESPF, WSPF, TVA, or a virtual memory operation requiring address translation.

### WRITE VIRTUAL PAGE (WVP)

The Write Virtual Page (WVP) command is a TOE on ERU 50 (32 H). This command loads one of the AM VPN entries. The entry loaded is the Least Recently Used VPN entry (i.e., the highest priority entry with its RR bit reset). The virtual page portion of the AM is loaded, left justified, from the PM bus. The format of the data transfer portion of the TOE is shown in Figure 5-9.



X — Don't Care

Bits 32 through 25 of the AM VPN are cleared if CA5 (Address Size) specifies 24 bit address size (i.e., CA5 = 0).

GIM5021

Figure 5-9 WVP Format

Upon execution of the WVP command, the corresponding Page Descriptor is loaded into the RAR and subsequently into the DDR. Upon completion of the WVP, the corresponding IR bit is set (i.e., this entry is invalidated; to validate see the WSPF command). The format of the DDR contents is shown in Figure 5-8.

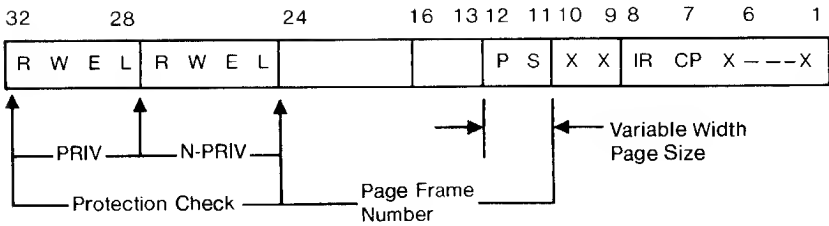
The DDR contents from this operation remain unaltered until a virtual address translation (attempt) occurs, or any of 3 commands is executed: TVA, WVP or ESPF.

### READ PAGE FRAME (RPF)

The Read Page Frame (RPF) command is a TIE operation on ERU 50 (32 H). This command transfers the contents of the DDR register to the PM bus. The format of the DDR contents is based on the last operation loading it. The format for a last-operation ESPF, WVP, or TVA command, is given in the individual command description. The format for a virtual operation is shown in the RRA command description section of this chapter.

### WRITE AND SET PAGE FRAME (WSPF)

The Write and Set Page Frame (WSPF) command is a TOE operation on ERU 51 (33 H). This command loads the Page Descriptor portion of an entry in the DAT. The entry written is the Least Recently Used Page Descriptor entry (i.e., highest priority entry with the RR bit reset). The Protection Check bits (both Privileged and Non-Privileged), the PFN, and the CP bit are loaded. The IR bit for that entry is reset (i.e., validated), and the RR bit is set. The format of the Page Descriptor load from the PM bus is shown in Figure 5-10.



R—Read, Protection Check

W—Write, Protection Check

E—Execute, Protection Check

L—Linkage, Protection Check

PS—The PS (page size) field is the least significant two bits contained in the PFN. Whether both (b12 and b11), b12, or neither bit is used to form the real address is a function of the page size specified. When one or more of these bits are not used, they are obtained from the VAR.

X—Don't Care

CP—Changed Page Bit

Note: If the CP bit is loaded as a "1", the Changed Page interrupt will not occur for this entry.

GIM5023

Figure 5-10 Page Descriptor Format

**Restriction —** The WSPF that accesses the sixteenth entry cannot be immediately followed by another WSPF, ESPF, TVA, or a virtual memory operation requiring address translation.

## CLEAR ASSOCIATIVE MEMORY (CAM)

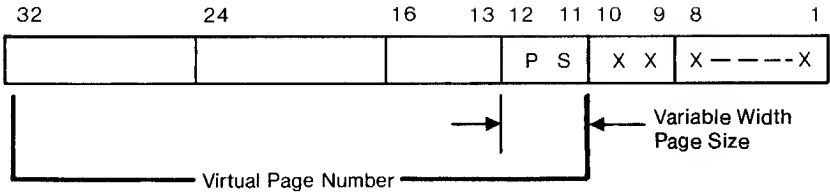
The Clear Associative Memory (CAM) command is a TIE operation on ERU 51 (33 H). This command resets all the RR bits. The data transfer portion of the CAM is not specified (i.e., is ignored).

The DAT is not invalidated by this operation (a virtual address can still have a successful translation).

## PURGE SELECTIVE (PS)

The Purge Selective (PS) command is a TOE operation on ERU 52 (34 H). The data portion of the PS command is compared with the VPN entries in the DAT. The PSR (Page Size Register) is used to determine which of the low order bits (b12 and b11) are used in the comparison (association) process. If the data portion is equal to any VPN entry, that entry is invalidated (i.e., the IR bit is set, and the RR bit is reset). If no equality is found, no action in the DAT occurs. The format of the PS data field is shown in Figure 5-11.

Address Translation Chip (ATC)



X — Don't Care

PS—The PS (page size) field is the least significant two bits of the data portion of the PS command. Whether both bits, b12 only, or neither bit is used in the comparison process is a function of the page size specified.

Page Size	Bit Used
4K	neither
2K	b12
1K	b12, b11 (both)

GIM5024A

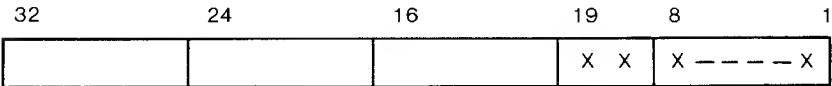
Figure 5-11 PS Data Field Format

Restrictions—

1. The PS command is a two cycle operation by the ATC. The ATC assures itself of the second cycle by activating the REQS line during the first cycle. This prevents any subsystem other than the processor from having access to the bus during the second cycle. The processor cannot follow the first command with any other operation that requires the ATC (i.e., no memory operations or ATC ERU operations) during the subsequent (second) cycle.
2. If CA5 (Address Size) is reset (i.e., specifying a 24 bit address), bits 32 through 25 of the PS command are not used in the comparisons and are treated as cleared bits. See the WVP command for the effect of CA5 on the AM VPN contents.

WRITE PURGE MASK (WPM)

The Write Purge Mask (WPM) command is a TOE operation on ERU 53 (35 H). This command loads the upper 22 bits of the Purge Mask Register (PMR). The format of the transfer and register is:



X — Don't Care

GIM5025

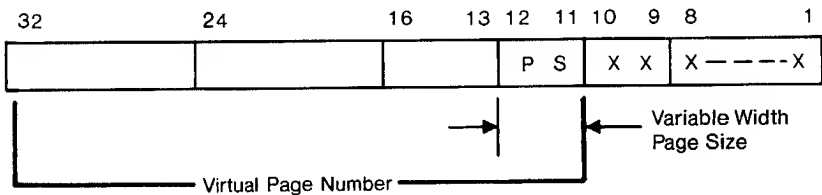
The PMR is used in conjunction with the PSM command.

## READ PURGE MASK (RPM)

The Read Purge Mask (RPM) command is a TIE operation on ERU53 (35 H). For a description of the format, see the WPM command.

## PURGE SELECTIVE WITH MASK (PSM)

The Purge Selective with Mask (PSM) command is a TOE operation on ERU 54 (36 H). The PMR (Purge Mask Register), in conjunction with the PSR (Page Size Register used to obtain the pertinent bits from the PMR), is used as a mask between the data portion of the PSM command and the VPN (Virtual Page Number) entries. Bits set in the mask inhibit association of corresponding VPN bits (i.e., these bits become “don’t care” bits in that their equality is guaranteed). If the PSM data, in conjunction with the mask, finds any VPN(s) equal, the entry is invalidated (the IR bit is set and the RR bit reset). If no equality is found, no action on the DAT occurs. The format of the PSM data field is shown in Figure 5-12.



X — Don't Care

PS—The PS (page size) field is the two least significant bits of the data portion of the PS command. Whether both bits, b12 only, or neither bit is used in the comparison process is a function of the page size specified.

Page Size	Bit Used
4K	neither
2K	b12
1K	b12,b11 (both)

GIM5026A

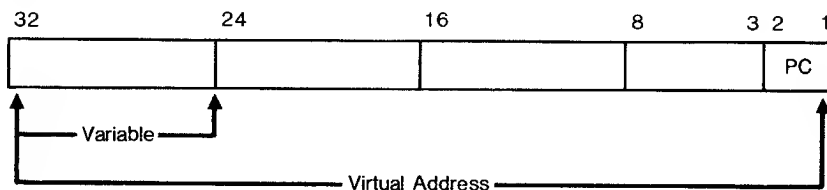
Figure 5-12 PSM Data Field Format

**Restrictions—**

1. The PSM command is a two cycle operation by the ATC. The ATC assures itself of the second cycle by activating the REQS line during the first cycle. This prevents any device other than the processor from having access to the bus during the second cycle. The processor cannot follow the first command with any other operation that requires the ATC (i.e., no memory operations or ATC ERU operations) during the subsequent (second) cycle.
2. If CA5 (Address Size) is reset (i.e., specifying 24 bit addresses) bits 32 through 25 of the PSM command are not used in the comparisons and are treated as cleared bits. See the WVP command for the effect of CA5 on the AM VPN contents.

**READ VIRTUAL ADDRESS (RVA)**

The Read Virtual Address (RVA) command is a TIE operation on ERU 54 (36 H). This command asserts the contents of the VARB onto the PM Bus. The VARB contains the last address for which a translation was attempted (i.e., the last virtual address or the data portion of a TOE operation on ERU 55, called the TVA command). If CA5 (Address Size) was reset, specifying 24 bit addresses, when the VAR was last loaded during translation, VARB bits 32-25 will be zeros. The format of the VARB contents asserted on the PM Bus is:



Variable — A function of CA5, as noted.

PC-Bits 2 and 1 are the access protection check code.

GIM5027A

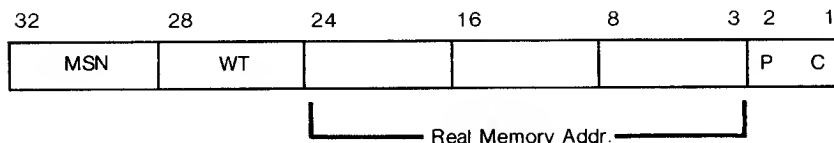
Figure 5-13 VARB Format

**READ REAL ADDRESS (RRA)**

The Read Real Address (RRA) command is a TIE operation on ERU 55 (37 H). The 32 bit content of the RAR is asserted onto the PM bus.

The content of the RAR is a function of the last real memory fetch operation, virtual memory operation, or ERU operation affecting it (i.e., ESPF, WVP, or TVA).

If the last operation affecting the RAR was a real memory fetch, a virtual equals real memory operation, a successful virtual memory address translation, or a successful Translate Virtual Address command, the RAR content format is as shown in Figure 5-14.



MSN — Most Significant Nibble

WT — Write Tags

PC — Protection Check

	Real	VER	VIRT	TVA
MSN	<u>PMBUS MSN</u>	<u>PMBUS MSN</u>	*	*
WT	<u>PMBUS28-25</u>	<u>PMWT0-3</u>	<u>PMWT0-3</u>	**
Real Addr.	<u>PMBUS24-03</u>	<u>PMBUS24-03</u>	***	***
PC	<u>PMBUS02,01</u>	<u>PMBUS02,01</u>	<u>PMBUS02,01</u>	<u>PMBUS02,01</u>

\*MSN is not loaded for this operation. The contents can be determined from the most recent of real or virtual equals real operation.

\*\*The WT is not loaded for this operation. The contents can be determined from the most recent of: real, virtual equals real or virtual memory operation.

\*\*\*PMBUS10-03 are directly loaded to bits 10-3. Bits 12 and 11 source is determined by the page size. Bits 24-13 are the page frame number directly corresponding to the successful match entry.

GIM5029

Figure 5-14      RAR Format

If the last operation affecting the RAR was an unsuccessful virtual address translation (memory or TVA), the RAR field definition is as shown in the TVA and VIRT columns of Figure 5-14.

If the last operation affecting the RAR was an ESPF or a WVP command, the RAR content format is the same as the DDR content format resulting from those commands.

**TRANSLATE VIRTUAL ADDRESS (TVA)**

The Translate Virtual Address (TVA) command is a TOE operation on ERU 55 (37 H). The 32 bit data portion of the TVA command (a virtual address) is loaded into the DAT logic and translation is attempted. This 32 bit “virtual address” is retained in the VARB until another address translation is attempted. No memory operation is initiated, regardless of the success or failure of the translation attempt. Note that CA9 (V.E.R. mode) has no effect on the TVA command.

A successful address comparison in the AM results in a translated (real) address being generated and loaded into the RAR (Real Address Register), and the RR bit being set for that entry.

An unsuccessful search in the AM results in the Dat No Match Interrupt bit being set in the I/T Array (I/TA 10), if enabled in the IMR. The RAR register format is the same as for a successful search, except that the real address bits are all asserted.

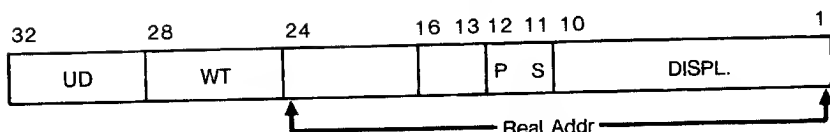
In either case no monitoring functions are performed, the CP bit is not affected, and no monitoring or CP interrupts are asserted. A protection check is performed using the two least significant bits of the data portion of the TVA command:

b2	b1	Protect Ck
0	0	read
0	1	write
1	1	execute
1	0	linkage

If translation is successful, the protection check is executed. If the protection check fails (indicating an illegal access), the DAT Access Violation bit (I/TA 11) is set in the I/T Array if enabled in the IMR.

If translation is successful, the corresponding Page Frame is loaded into the RAR and then into the DDR. If the translation is unsuccessful, the contents in the RAR and DDR are undefined. For successful translation, the format of the DDR contents is:





- UD—Undefined. These four bits are not loaded for this operation. The contents can be determined from the most recent of: real memory fetch, virtual equals real operation, ESPF or WSPF.
- WT—Write Tags. These four bits are not loaded for this operation. The contents can be determined from the most recent of: real, virtual equals real, or virtual memory operation.
- Real Addr—Bits 10-1 are loaded from the PM Bus. Bits 12 and 11 source is determined by the page size. Bits 24-13 are the page frame number directly corresponding to the table entry that matched.

GIM5028A

Figure 5-15 DDR Format, TVA Command

The DDR contents from this operation remain unaltered until a virtual address translation is attempted, or a TVA, WVP, or ESPF command is executed.

### Restrictions—

1. The TVA command is a two cycle operation for the ATC. The ATC assures itself of the second cycle by activating the REQS line during the first cycle. This prevents any device other than the processor from having access to the bus during the second cycle. The processor cannot follow the first command with any other operation that requires the ATC (i.e., no memory operations or ATC ERU operations) during the subsequent (second) cycle.
2. If CA5 (Address Size) is reset (i.e., specifying 24 bit addresses) bits 32 through 25 of the TVA command are not used in the comparisons and are treated as cleared bits. See the WVP command for the effect of CA5 on the AM VPN contents.

### ATC STATE OPERATION

ATC memory operations are controlled by state sequencing, shown in Figure 5-16. Pertinent bus control inputs are latched and sampled at the end of each X0 clock interval for use in determining the ATC state during the following clock cycle beginning at X0. Each state extends through one cycle (X0 interval to X0 interval), although certain states may be repeated pending the assertion or de-assertion of pertinent bus control signals (see DIE wait loops, Figure 5-16, states 1, 2, and 3).

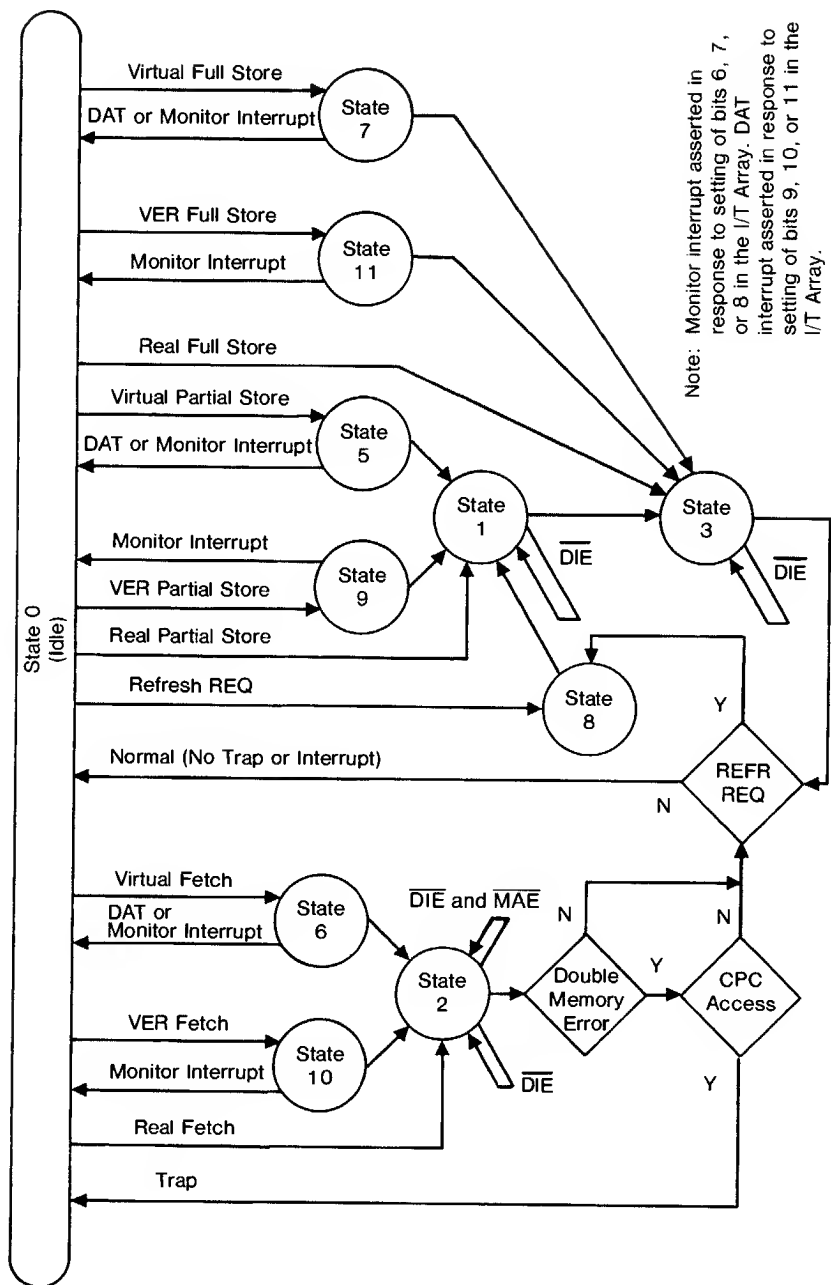


Figure 5-16 ATC State Sequencing

## STATE FLOW

Each "normal" memory operation begins in State 0 and ends in State 0. Two conditions during memory operations, however, will alter this normal flow:

1. If an ATC interrupt occurs (any of I/T Array bits 6-11 sets) the memory operation is aborted, and the ATC returns to State 0.
2. If a memory refresh condition (REF REQ) occurs during a memory operation and no trap is pending, the ATC enters a memory refresh sequence immediately following the memory operation before returning to State 0.

Table 5-9 shows the names of the states. The names are descriptive of ATC activity in each state.

State Number	State Name
0	Idle (No Memory Operation in Progress)
1	Real Partial Store
2	Real Fetch
3	Real Full Store
4	Not Used
5	Virtual Partial Store
6	Virtual Fetch
7	Virtual Full Store
8	Refresh
9	VER* Partial Store
10	VER* Fetch
11	VER* Full Store

\*VER — Virtual Equals Real

GIITE5032

Table 5-9 ATC State Names

## SPECIAL ATC CONSIDERATIONS

The following paragraphs describe ATC characteristics which should be considered when using the ATC.

### PM BUS CONTENTION

The ATC PM Bus request for memory access (real, virtual, and refresh), REQ0, has highest bus request priority and must be recognized by all other devices on the PM Bus. No devices on the PM Bus except the ATC and memory interface logic must drive the bus during any cycle for which REQ0 was asserted.

### REFRESH

The refresh operation is disabled if any of the following conditions are met: INH or PMRST input is active (low), the double bit

memory error trap bit is set, or a TOE operation to the Syndrome Register has not been followed by a TIE operation to the Syndrome Register.

### TIME-OF-DAY

A TOE operation to the TOD must not alter bits 10 through 1 of the TOD register if these ten bits are utilized as the refresh row address.

TOD interrupts are input active for only four microseconds, based on a 250KHz clock. Therefore, the time elapsed between a TIE to the I/T Array and a TOE to the I/T Array should be less than four microseconds to guarantee not losing a TOD interrupt.

### BUS INTERRUPT REGISTER INTERRUPTS

The BIN interrupt must be cleared as follows: read I/T Array, read BIN, then write I/T Array. All three operations must be executed, with the write operation last, to guarantee proper BIN interrupt operation.

### ASSOCIATIVE MEMORY COMMAND SEQUENCING

The ESPF and WSPF commands result in the RR bit of a Page Descriptor entry in the DAT being set. When either of these commands results in the sixteenth Page Descriptor entry having its RR bit set, the next sequential operation can not be a WSPF, ESPF, or TVA command, or a virtual memory operation requiring translation.

The TVA, PS, and PSM commands each require two cycles to complete. The ATC asserts REQS during the first cycle of these three commands. This allows the processor, and no other device, use of the PM Bus during the second cycle, with the restriction that the processor must not execute an ATC related operation during this second cycle.

### MONITOR OPERATIONS

Control Array bits 3 and 8 are mutually exclusive, resulting in monitor operation being inoperative if both bits are simultaneously set. Control Array bits 2 and 4 are mutually exclusive by operation, resulting in indeterminate results if both bits are simultaneously set.

### ECC DISABLE

Control Array bit 7 disables data bit correction and memory error reporting (i.e.,  $\overline{\text{MDEE}}$ ,  $\overline{\text{MEMERR}}$ , and  $\overline{\text{TRAP}}$ ). Partial stores (including refresh operation) with CA7 set fetch in data, do not correct or report errors, and write back to memory with syndrome bits generated for the write back data.

## ECC GENERATE/SYNDROME REGISTER

A TOE operation to the Syndrome Register forces the ATC to cease ECC generation and to disable refresh operations. A TIE operation from the Syndrome Register releases the ATC hardware to allow ECC generation and allow refresh operations. During the period between the TOE and the TIE operation to the Syndrome Register, memory store operations use the contents of the Syndrome Register for store check bits (Syndrome). The contents of the Syndrome Register following a TOE operation are the most recent of either the TOE operation data to the Syndrome Register, or the syndrome bits returned during a memory fetch.

## REAL ADDRESS REGISTER BYTE/DESCRIPTOR DATA REGISTER

The RAR lower 10 to 12 bits and byte 0 contents may or may not be directly related to the most recent use of the RAR. Programmer use of the lower 10 to 12 bits or byte 0 contents (via a TIE operation) should be restricted to only valid bit fields. Since the DDR is directly loaded from the RAR, the same restriction applies to DDR accesses.

## 24/32 BIT OPERATIONS

Associative Memory operations in the 24 bit mode automatically clear byte 0 during loading of AM entries and during the address association (match) process. The monitor operations disregard byte 0 in the 24 bit mode.

## ASSOCIATIVE MEMORY RESULTS

Associative memory operation (address translation) results should be read from the DDR rather than the RAR, since the RAR contents can be altered by an I/O device on the PM Bus.

## TIMING CYCLE DESCRIPTIONS

The following are detailed timing cycle descriptions for real memory operations, virtual memory operations, and refresh operations.

## REAL MEMORY OPERATIONS

Real memory operations are identified by an asserted  $\overline{\text{MAE}}$  during X0. The real address is asserted on the PM Bus during the same X0, and is latched into the ATC RAR register if the operation is a fetch. The ATC examines the write tags (bits 28-25) to determine the type of real memory operation to be performed:

- $\overline{\text{PMBUS28-25}}$ —All active implies a full store  
 —Some but not all active implies a partial store  
 —All inactive implies a fetch

## Real Full Store

The real full store is a minimum 2-cycle operation.

Cyc. No.	Clk.	Operation
1	X0	Activate $\overline{REQ0}$
	X1	Output $\overline{REQ0}$ active Load data into MD/PD (PD) and generate syndrome bits
2	X0	Output data and syndrome bits to memory and hold stable until after $\overline{DIE}$ goes active
	X1	Continue holding data and syndrome stable to end of clock

GIMTE5035

Table 5-10 Real Full Store

## Real Partial Store

The real partial store (read-modify-write) is a minimum 3-cycle operation.

Cyc. No.	Clk.	Operation
1	X0	Activate $\overline{REQ0}$
	X1	Hold $\overline{REQ0}$ active Load data into MD/PD (PD) register
2	X0	Activate $\overline{REQ0}$ Latch data from memory into MD/PD (MD) if $\overline{DIE}$ is active Check and correct memory data
	X1	Hold $\overline{REQ0}$ active Concatenate PD with MD for bytes being written Generate syndrome bits on new data
3	X0	Output new data and syndrome bits to memory and hold stable until after $\overline{DIE}$ goes active
	X1	Continue holding data and syndrome bits stable to end of clock

GIMTE5036

Table 5-11 Real Partial Store

## Real Fetch

The real fetch is a minimum 2-cycle operation.

Cyc. No.	Clk.	Operation
1	X0	Load real address into RAR
	X1	Activate $\overline{REQ0}$ Hold $\overline{REQ0}$ active
2	X0	Latch data from memory into MD/PD (MD) if $\overline{DIE}$ is active Check and correct data
	X1	Output the checked and corrected data to requester; if there was an uncorrectable error, activate $\overline{MDEE}$ and, if the fetch was initiated by the processor, activate $\overline{TRAP}$ .

GIMTE5037

Table 5-12 Real Fetch

## VIRTUAL MEMORY OPERATIONS

Virtual memory operations are identified by a CPC-asserted  $\overline{PVT}$  during X0. During the same X0 the virtual address is asserted on the PM Bus. The ATC clocks the virtual address into its VAR at the end of that X0 and starts the translation process. The ATC examines the  $\overline{PMWT0-3}$  lines to determine the type of memory operation to be performed.

- $\overline{PMWT0-3}$ —All active implies a full store  
 —Some but not all active implies a partial store  
 —All inactive implies a fetch

### Virtual Full Store (CA9=0)

The virtual full store is a minimum 3-cycle operation (i.e., a translation cycle and a real full store).

Cyc. No.	Clk.	Operation
1	X0	Load Virtual Address into the VAR Activate $\overline{REQ0}$ Start translation
	X1	Hold $\overline{REQ0}$ active, load VAR into VARB Load data into MD/PD (PD) register and generate syndrome bits Load real address from translation into RAR
2	X0	Activate $\overline{MAE}$ Activate $\overline{REQ0}$ Send address and write tags to memory
	X1	Hold $\overline{REQ0}$ active
3	X0	Send data and syndrome bits to memory and hold stable until cycle after $\overline{DIE}$ goes active ( $\overline{DIE}$ going active indicates that the store operation will be completed in the next cycle)
	X1	Continue holding data and syndrome bits stable to end of clock

GIMTE5038

Table 5-13 Virtual Full Store (CA9=0)

**Virtual Partial Store (CA9=0)**

The virtual partial store is a minimum 4-cycle operation (i.e., a translation cycle and a real partial store).

Cyc. No.	Clk.	Operation
1	X0	Load virtual address into VAR Activate $\overline{REQ0}$ Start Translation
	X1	Hold $\overline{REQ0}$ active, load VAR into VARB Load data into MD/PD (PD) register Load real address from translation into RAR
2	X0	Activate $\overline{MAE}$ Activate $\overline{REQ0}$ Send real address and write tags to memory
	X1	Hold $\overline{REQ0}$ active
3	X0	Latch memory data into MD/PD (MD) if $\overline{DIE}$ is active Check and correct data Activate $\overline{REQ0}$
	X1	Hold $\overline{REQ0}$ active Concatenate PD with MD for bytes being written Generate syndrome bits on new data
4	X0	Output new data and syndrome bits to memory and hold stable until after $\overline{DIE}$ goes active (it is assumed that the Memory Interface has the address from the second X0)
	X1	Continue to hold data and syndrome bits stable to end of clock

GIMTE5039

Table 5-14 Virtual Partial Store (CA9=0)

**Virtual Fetch (CA9=0)**

The virtual fetch is a minimum 3-cycle operation (i.e., a translation cycle and a real fetch).

Cyc. No.	Clk.	Operation
1	X0	Load virtual address into VAR Activate $\overline{REQ0}$ Start Translation
	X1	Hold $\overline{REQ0}$ active, load VAR into VARB Load real address from translation into RAR
2	X0	Activate $\overline{MAE}$ Send real address and write tags to memory Activate $\overline{REQ0}$
	X1	Hold $\overline{REQ0}$ active
3	X0	Latch data from memory into MD/PD (MD) if $\overline{DIE}$ is active Check and correct data
	X1	Output checked and corrected data to requester, if there was an uncorrectable error, activate $\overline{MDEE}$ and if it was a processor operation activate $\overline{TRAP}$

Table 5-15 Virtual Fetch (CA9=0)

GIMTE5040



**Virtual Full Store (CA9 = 1)**

Cyc. No.	Clk.	Operation
1	X0	Load virtual address into VAR Activate REQ0
	X1	Hold REQ0 active, load VAR into VARB and RAR Load data into MD/PD (PD) register and generate syndrome
2		Same as for CA9=0
3		Same as for CA9=0

GIMTE5041

Table 5-16 Virtual Full Store (CA9 = 1)

**Virtual Partial Store (CA9 = 1)**

Cyc. No.	Clk.	Operation
1	X0	Load virtual address into VAR Activate REQ0
	X1	Hold REQ0 active, load VAR into VARB and RAR Load data into MD/PD (PD) register and generate syndrome
2		Same as for CA9=0
3		Same as for CA9=0
4		Same as for CA9=0

Table 5-17 Virtual Partial Store (CA9=1)

**Virtual Fetch (CA9 = 1)**

Cyc. No.	Clk.	Operation
1	X0	Load virtual address into VAR Activate REQ0
	X1	Hold REQ0 active, load VAR into VARB and RAR
2		Same as for CA9=0
3		Same as for CA9=0

GIMTE5043

Table 5-18 Virtual Fetch (CA9 = 1)

## REFRESH OPERATION

When a memory refresh is required and other conditions permit, the ATC asserts  $\overline{REQ0}$  during X0 to secure the bus for the next cycle. If the ATC is currently doing a memory operation, the refresh state is entered upon completion of the memory operation if no traps are pending. If the ATC is idle, if there are no requests for memory or ATC ERU operations, and if no traps are pending, the refresh state is entered. If a TOE operation on ERU 49 (called WSB) has been executed and a TIE on ERU 49 (called RSB) has not been executed, the ATC will not recognize the need for refresh; however, the condition is held for 16 microseconds. Refresh is a minimum 4-cycle operation:

Cyc. No.	Clk.	Operation
1	X0 X1	Activate $\overline{REQ0}$ Hold $\overline{REQ0}$ active
2	X0  X1	Activate $\overline{REQ0}$ Activate $\overline{MAE}$ Send refresh address to memory with $\overline{PMBUS24}$ active (low) and the write tags inactive Hold $\overline{REQ0}$ active Reset the refresh request signal
3	X0  X1	Activate $\overline{REQ0}$ Latch fetched data into MD/PD (MD) if $\overline{DIE}$ is active; check syndrome bits and correct single bit error Hold $\overline{REQ0}$ active If uncorrectable error detected activate $\overline{MDEE}$
4	X0  X1	Output data and syndrome bits to memory and hold both stable until after $\overline{DIE}$ goes active Continue holding data and syndrome bits stable to end of clock

GIMTE5044

Table 5-19 Refresh

## CHAPTER VI MICROINSTRUCTION SET

### CONTENTS

Microinstruction Set Format . . . . .	6-1
L Field Function . . . . .	6-1
K Field Function . . . . .	6-1
J Field Function . . . . .	6-2
I Field Function . . . . .	6-2
H Field Functions . . . . .	6-2
G Field Functions . . . . .	6-2
Instruction Nomenclature . . . . .	6-2
Instruction Operands . . . . .	6-4
Full Word Operands . . . . .	6-4
Halfword Operands . . . . .	6-5
Byte Operands . . . . .	6-5
Condition Selector . . . . .	6-6
Field Operands . . . . .	6-6
Single Field Operand Instructions . . . . .	6-7
Multiple Field Operand Instructions . . . . .	6-7
Literal Operands . . . . .	6-8
Four Bit Literal . . . . .	6-8
Eight Bit Literal . . . . .	6-8
Sixteen Bit Literal . . . . .	6-9
Digit Operands . . . . .	6-9
Instruction Descriptions . . . . .	6-9
Memory Instructions . . . . .	6-9
Instruction Index by Function . . . . .	6-9
Instruction Index by Op Code . . . . .	6-15
Instruction Index by Mnemonic . . . . .	6-20

## CHAPTER VI

# MICROINSTRUCTION SET

The NCR/32-000 (CPC) has been developed to provide a high degree of performance using 179 instructions and variants organized into the following general categories:

- Memory instructions
- Transfer instructions
- Field (string) instructions
- Arithmetic instructions
- Branch instructions
- Special emulation instructions
- Special instructions

### MICROINSTRUCTION SET FORMAT

The CPC instruction set is a 16 or 32 bit instruction divided into the following fields:



GIM4003

#### L Field Function

- Used as a jump literal address in several branch instructions.
- Used to load literals into the RSU.

#### K Field Function

- Used as an RSU address in word or halfword instructions.
- Used as an RSU byte address for the four RSU registers 0 through 3.
- Used as a MARS register addresses—all MARS registers are located in the RSU.
- Used as a Literal.

**J Field Function**

- Used as an RSU address in word and halfword instructions.
- Used as an RSU byte address for RSU registers 0 through 3.
- Used as a MARS register address—all MARS registers are located in the RSU.
- Used as a Jump Register Address.
- Used as a Literal.

**I Field Function**

- Used as part of the operation code in some of the microcommands.
- Used as part of the address field for external registers and Main Memory locations (scratch pad).
- Used as a literal.

**H Field Functions**

- Used as part of the operation code in some microcommands.
- Used as a literal.

**G Field Functions**

- Used as part or all of the operation code.

**INSTRUCTION NOMENCLATURE**

The nomenclature described here is used in the instruction set summary.

CR	Control Register
(CR)	The contents of the Control Register
ISU	Instruction Storage Unit
(ISU)	The contents of the ISU
JRJ	The Jump Register specified by the J field of the instruction
RSU	Register Storage Unit, 16 word (32-bit) registers
RJ	The RSU Register specified by the J Field of an instruction

(RJ)	The contents of the RSU location specified by J
RK	The RSU Register specified by the K field of an instruction
(RK)	The contents of the RSU location specified by K
RKO	The RSU Register specified by the K field of an instruction containing an odd number
RKE	The RSU Register specified by the K field of an instruction containing an even number
R13	The MARS data register (RSU) used for data write operation
R9	One of the MARS data registers used for fetching data fields
R11	One of the MARS data registers used for fetching data fields
R15	The MARS Data Register used to hold Virtual Instructions
XIJ	The external register specified by the IJ literal
(XIJ)	The contents of the External Register specified by the IJ literal
L	A 16 bit literal
(C)	The contents of the Carry Indicator in the Indicator Array
M#OF	MARS Overflow Flag which indicates that the MARS Byte Pointers have crossed the word boundary
M#B#	The Byte Pointer corresponding to the particular MARS unit
MARS	Memory Assist Register Set
Word	32 bit field, basic CPC memory unit

## *MICROINSTRUCTION SET*

Halfword	16 bit container, either most significant or least significant 16 bits of a word
Byte	8 bit field, four per word
Digit	4 bit field, either most significant or least significant 4 bits of a byte
CPC	Central Processor Chip, NCR/32-000
ATC	Address Translation Chip, NCR/32-010
EAC	Extended Arithmetic Chip, NCR/32-020
PM BUS	Processor-Memory Bus, link between CPC, ATC, EAC and Main Memory
Packed	Two BCD digits within a byte. Arithmetic operations generate a carry from the least significant digit to the most significant digit and from the most significant digit to the Carry Indicator
C.S.A.	Control Store Address (ISU Address)

## **INSTRUCTION OPERANDS**

Instructions utilize seven types of operands:

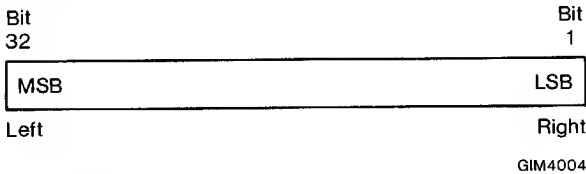
- Full Word Operands
- Half Word Operands
- Bytes
- Test MASK Operands
- Field Operands
- Literal Operands
- Digit Operands

### **Full Word Operands**

Full Word Operands are 32 bits in length. In the CPC there are 32-bit paths between:

- RSU and Main Memory (data)
- RSU and RSU (word transfer)
- RSU and ALU (word arithmetic and logic)
- RSU and External Registers
- RSU and Scratch Pad (local memory data)

The Full Word format is shown below:

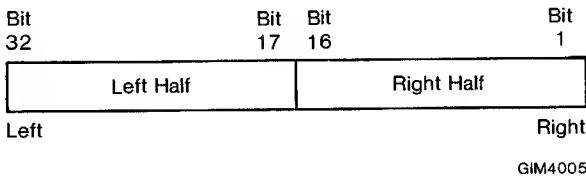


### Halfword Operands

Halfword operands are 16 bits in length. The number of halfword operands has been intentionally minimized to simplify ALU hardware. There are 16 bit paths between:

- RSU and JRJ
- CR and JRJ
- RSU and Tally Register
- RSU and RSU
- RSU and Setup Registers
- RSU and some ERUs

The halfword format is shown below:



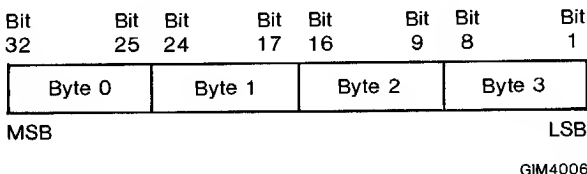
### Byte Operands

Byte operands are 8 bits in length. There are 8-bit paths between:

- RSU and RSU
- RSU and ALU

Bytes are specified by the two lower order bits of the J and K fields of byte operations. The bytes in RSU 0 through 3 are explicitly addressable.

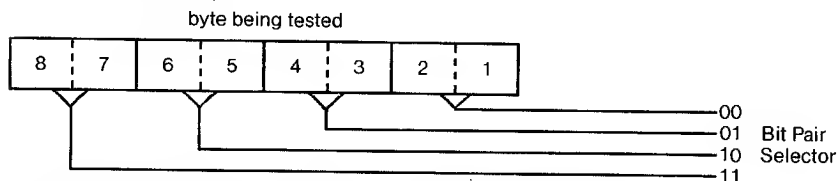
The bytes within an RSU word are located as shown below:





## Condition Selector

For some Conditional Jump and Conditional Skip instructions, the J or K field contains a Condition Selector. This Condition Selector consists of two parts: the Bit Pair Selector and the Bit Pair Mask. **Bit Pair Selector**—The function of the Bit Pair Selector is to specify a pair of bit positions to be tested (from the byte indicated by the microinstruction).



The specified bits are referred to as the Test Bits. If, for example, the Bit Pair Selector is “00”, the Test Bits are bits 1 and 2 of the byte. A Bit Pair Selector of “10” specifies bits 5 and 6 as the Test Bits.

**Bit Pair Mask**—The function of the Bit Pair Mask is to specify which of the Test Bits affect the outcome of the test. The particular microinstruction being used determines whether a one or a zero value in the Test Bits will satisfy the test.

Bit Pair	00	—	Either bit (or both) can satisfy the test.
	01	—	Right bit must satisfy the test.
Mask	10	—	Left bit must satisfy the test.
	11	—	Both bits must satisfy the test.

**Examples**—These examples show how the Bit Pair Selector and the Bit Pair Mask of the Condition Selector work together.

### Example 1

Bit Pair Selector = 01

Bit Pair Mask = 10

### Example 2

Bit Pair Selector = 11

Bit Pair Mask = 00

For example 1, bit 4 of the byte being tested must satisfy the test condition because: 1) Bit Pair Selector 01 specifies Test Bits 4 and 3; and 2) Bit Pair Mask 10 specifies the left bit of those two, or bit 4. Example 2 indicates that either bit 8 or bit 7 of the byte being tested must satisfy the test condition. Bit Pair Selector 11 specifies Test Bits 8 and 7; Bit Pair Mask 00 specifies either of those bits, 8 or 7.

## Field Operands

Field operands are from 1 to (64K-1) bytes in length. The fields are located in Main Memory. The processor works on one, two or three fields at a time. A field is specified by the contents of a MARS register and the contents of a Tally Register. The MARS registers are specified by the J and K fields. The Store MARS is always MARS6 (implied by the instruction). All fields handled by the hardware must be of equal length.

The MARS operation is specified by the instruction operation code. Fields are processed one byte at a time under hardware control. The field instructions will not move in the pipeline until a word boundary is crossed or the Tally Register equals zero. The data is transferred between Main Memory and the RSU (four bytes at a time) under Firmware control. While in the RSU, the bytes are addressed by the two low-order bits of the corresponding MARS register. Thus, the instruction selects the RSU word and the MARS selects the byte within the word. The Tally is decremented as each byte is processed.

When Tally = 0, the field operation is complete, the pipeline is advanced to the next instruction, and the Firmware sends a partial store to Main Memory if required.

If Tally = 0 when the Field command is entered, then the specified transfer is not executed.

The MARS Data Registers to be used in a Field instruction must be pre-loaded before entering the Field operation. For Arithmetic Field instructions, the Carry Indicator must be pre-set to the proper value.

The Carry and the BCD indicators (both PBCD and UBCD) are the only Indicators that respond through the individual cycles of the Field instructions. The Carry (I4) must be initialized by Firmware prior to the first execution of an AF, APDF, AUDF, SF, SPDF or SUDF instruction. The Carry is chained automatically throughout the subsequent execution cycles. I4 is initialized to a zero (via an RIZ instruction) for AF, APDF and AUDF. I4 is initialized to a one (via an SCO instruction) for SF, SPDF and SUDF.

### Single Field Operand Instructions —

The following is an example of a firmware flow using a single field operand instruction:

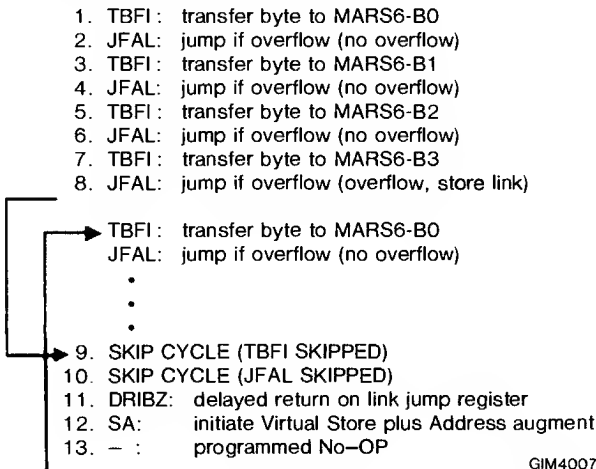


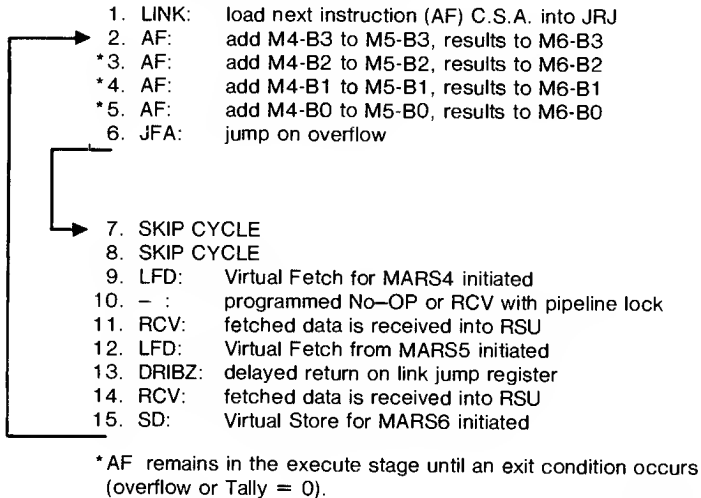
Figure 6-1 Single Field Operand Microcode

The Byte to Field, Halfword to Field, Field to Byte and Field to Halfword instructions are all single field operand instructions. Only one of the instruction operands is a Field operand. These instructions execute in a single cycle.

**Multiple Field Operand Instructions** — The Transfer Field instructions, the Boolean Field instructions and the Arithmetic Field instructions are multiple field operand (two or three field) instructions. These instructions execute in one to four cycles depending upon whether a MARS byte pointer crosses the word boundary or the Tally Register decrements to zero.

These instructions hold in the execute stage of the pipeline until an exit condition occurs.

The following is an example of a firmware flow using a multiple field operand instruction where all fields are aligned:



GIM4008

Figure 6-2 Multiple Field Operand Microcode Flow

## Literal Operands

Literal operands are 4, 8 or 16 bits in length and are included in the instruction. Literal operands are typically used as follows:

### Four Bit Literal

- Used as a constant in arithmetic functions

### Eight Bit Literal

- A constant to be loaded into RSU
- An offset address for CR literal jump
- A literal value to be loaded into Tally or transferred to the EAC

## Sixteen Bit Literal

- An absolute jump address
- A literal to load into RSU

## Digit Operands

Digit operands are 4 bits in length. There are 4-bit paths between RSU's only. Digit operands are specified within the four byte-addressable RSU's.

## INSTRUCTION DESCRIPTIONS

The CPC instructions are grouped into the following categories:

- Memory Instructions
- Transfer Instructions
- Logical Instructions
- Arithmetic Instructions
- Jump Instructions
- Miscellaneous Instructions

Unless otherwise indicated, all instructions are single cycle instructions.

## Memory Instructions

All memory instructions transfer information over the Processor-Memory Bus. The memory operations are either virtual or real.

**Important:** Virtual Store instructions should never be immediately followed by a Virtual Memory instruction. If a DAT Fault occurs during the execution of a Virtual Store instruction, and if the next instruction initiates a Virtual operation, then the Virtual Address from the faulted operation will be over-written by the Virtual Address from the subsequent operation.

Virtual Fetch instructions may be immediately followed by a Virtual Memory instruction (DAT Faults do occur during the Fetch sequence). However, if a Memory Error Trap occurs, the Virtual Address from the first instruction will be over-written by the Virtual Address from the subsequent operation. If the Virtual Address must be saved for a Memory Error Trap routine, then the same restriction as for Virtual Store instructions must be applied to the Virtual Fetches.

## INSTRUCTION INDEX BY FUNCTION

Tables 6-1 to 6-6 show the CPC instructions grouped by function, and listed in ascending op code order within each group.

TABLE 6-1—MEMORY INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
02	MEMORY REFERENCE RETRY	MRR
03	FETCH REAL	FR
04	FETCH	F
05	LOAD, FETCH, AND AUGMENT	LFA
06	LOAD, FETCH AND AUGMENT LINKAGE	LFAL
07	LOAD, FETCH AND DECREMENT	LFD
08-0B	FETCH LITERAL	FL
14	STORE REAL	SR
15	STORE	S
16	STORE AND AUGMENT	SA
17	STORE AND DECREMENT	SD
18-1B	STORE LITERAL	SL

TABLE 6-2—TRANSFER INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
00-01	TRANSFER IN EXTERNAL (32-63)	TIE
0C-0F	TRANSFER IN PORT (64-127)	TIP
10-11	TRANSFER OUT EXTERNAL (32-63)	TOE
1C-1F	TRANSFER OUT PORT (64-127)	TOP
20-21	TRANSFER IN INTERNAL (0-31)	TII
22	TRANSFER FIELD TO LEFT HALFWORD DECREMENT	TFLHD
23	TRANSFER FIELD TO LEFT HALFWORD INCREMENT	TFLHI
24	TRANSFER FIELD TO RIGHT HALFWORD DECREMENT	TFRHD
25	TRANSFER FIELD TO RIGHT HALFWORD INCREMENT	TFRHI
26	TRANSFER FIELD TO FIELD DECREMENT	TFFD
27	TRANSFER FIELD TO FIELD INCREMENT	TFFI
28	TRANSFER FIELD TO BYTE DECREMENT	TFBD
29	TRANSFER FIELD TO BYTE INCREMENT	TFBI
2A	TRANSFER FIELD TO BYTE DECREMENT NO TALLY	TFBDN
2B	TRANSFER FIELD TO BYTE	TFB
30-31	TRANSFER OUT INTERNAL (0-31)	TOI
3F	TRANSFER FIELD TO BYTE INCREMENT NO TALLY	TFBIN
51	TRANSFER WORD	TW
58	LOAD TALLY RIGHT CLEAR LEFT	LTRC
59	LOAD BYTE	LB
5A	LOAD LEFT DIGIT	LLD

TABLE 6-2—TRANSFER INSTRUCTIONS (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
5B	LOAD RIGHT DIGIT	LRD
5C	LOAD RIGHT HALFWORD	LRH
5D	LOAD RIGHT HALFWORD CLEAR LEFT	LRHC
A0	SHIFT WORD ARITHMETIC RIGHT	SWAR
A1	SHIFT WORD LOGICAL LEFT	SWLL
A2	SHIFT WORD LOGICAL LEFT WITH CARRY	SWLLC
A3	SHIFT WORD LOGICAL RIGHT	SWLR
A4	SHIFT WORD LOGICAL RIGHT WITH CARRY	SWLRC
A5	SHIFT WORD CIRCULAR LEFT	SWCL
A6	TRANSFER LEFT HALFWORD TO FIELD DECREMENT	TLHFD
A7	TRANSFER LEFT HALFWORD TO FIELD INCREMENT	TLHFI
A8	TRANSFER RIGHT HALFWORD TO FIELD DECREMENT	TRHFD
A9	TRANSFER RIGHT HALFWORD TO FIELD INCREMENT	TRHFI
AA	TRANSFER LEFT HALFWORD TO LEFT HALFWORD	TLHLH
AB	TRANSFER LEFT HALFWORD TO RIGHT HALFWORD	TLHRH
AC	TRANSFER RIGHT HALFWORD TO LEFT HALFWORD	TRHLH
AD	TRANSFER RIGHT HALFWORD TO RIGHT HALFWORD	TRHRH
BB	TRANSFER BYTE TO FIELD INCREMENT NO TALLY	TBFIN
BC	TRANSFER BYTE TO FIELD DECREMENT	TBFD
BD	TRANSFER BYTE TO FIELD INCREMENT	TBFI
BE	TRANSFER BYTE TO FIELD DECREMENT NO TALLY	TBFTN
BF	TRANSFER BYTE TO FIELD	TBF
CD	TRANSFER BYTE	TB
D2	TRANSFER LEFT DIGIT TO LEFT DIGIT	TLDL
D3	TRANSFER LEFT DIGIT TO RIGHT DIGIT	TLDRD
D4	TRANSFER RIGHT DIGIT TO LEFT DIGIT	TRDL
D5	TRANSFER RIGHT DIGIT TO RIGHT DIGIT	TRDRD

TABLE 6-3—LOGICAL INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
42	BOOLEAN AND FIELD	BAF
43	BOOLEAN OR FIELD	BOF
44	BOOLEAN EOR FIELD	BEF
4E	BOOLEAN AND WORD	BAW
4F	BOOLEAN OR WORD	BOW
50	BOOLEAN EOR WORD	BEW
5E	BOOLEAN INVERT WORD	BIW
CA	BOOLEAN AND BYTE	BAB
CB	BOOLEAN OR BYTE	BOB
CC	BOOLEAN EOR BYTE	BEB
D8	BOOLEAN AND LEFT DIGIT	BALD
D9	BOOLEAN AND RIGHT DIGIT	BARD
DA	BOOLEAN OR LEFT DIGIT	BOLD
DB	BOOLEAN OR RIGHT DIGIT	BORD
DC	BOOLEAN EOR LEFT DIGIT	BELD
DD	BOOLEAN EOR RIGHT DIGIT	BERD
DF	BOOLEAN INVERT BYTE	BIB

TABLE 6-4—ARITHMETIC INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
40	ADD FIELD	AF
41	SUBTRACT FIELD	SF
45	COMPARE FIELD UNSIGNED	CFU
46	ADD PACKED DECIMAL FIELD	APDF
47	SUBTRACT PACKED DECIMAL FIELD	SPDF
48	ADD UNPACKED DECIMAL FIELD	AUDF
49	SUBTRACT UNPACKED DECIMAL FIELD	SUDF
4A	ADD WORD	AW
4B	SUBTRACT WORD	SW
4C	ADD WORD WITH CARRY	AWC
4D	SUBTRACT WORD WITH CARRY	SWC
52	COMPARE WORD SIGNED	CWS
53	COMPARE WORD UNSIGNED	CWU
57	COMPARE BYTE TO FIELD UNSIGNED	CBFU
93	ADD CR TO LITERAL	ACRL
9C	ADD WORD WITH LITERAL NO INDICATOR CHANGE	AWLNI
9D	SUBTRACT WORD LITERAL NO INDICATOR CHANGE	SWLNI
AE	ADD WORD WITH LITERAL	AWL
AF	SUBTRACT WORD WITH LITERAL	SWL

TABLE 6-4—ARITHMETIC INSTRUCTIONS (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
C0	COMPARE BYTE SIGNED	CBS
C1	COMPARE BYTE UNSIGNED	CBU
C6	ADD BYTE	AB
C7	SUBTRACT BYTE	SB
C8	ADD BYTE WITH CARRY	ABC
C9	SUBTRACT BYTE WITH CARRY	SBC
CE	ADD PACKED DECIMAL BYTE	APDB
CF	SUBTRACT PACKED DECIMAL BYTE	SPDB
D0	ADD PACKED DECIMAL BYTE WITH CARRY	APDBC
D1	SUBTRACT PACKED DECIMAL BYTE WITH CARRY	SPDBC
DE	SUBTRACT BYTE LITERAL	SBL

TABLE 6-5—JUMP INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
2C	JUMP ON REGISTER	JOR
2D	DELAYED JUMP ON REGISTER	DJOR
32	IBM SETUP JUMP A	JMPIA
33	IBM SETUP JUMP B	JMPIB
34	IBM SETUP JUMP C	JMPIC
35	NVM SETUP JUMP A	JMPNA
36	NVM SETUP JUMP B	JMPNB
37	NVM SETUP JUMP C	JMPNC
38	VRX SETUP JUMP A	JMPVA
39	VRX SETUP JUMP B	JMPVB
3A	VRX SETUP JUMP C	JMPVC
3B	NVM DESCRIPTOR JUMP	JMPD
3C	JUMP ON PMBUS NEGATIVE	JPMBN
3E	DELAYED JUMP ON PMBUS NEGATIVE	DJPMBN
60	SKIP ON REGISTER BYTE 3 ONES	SRB30
62	DELAYED JUMP ON INDICATOR BIT PAIR ONES	DJIBO
63	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS	DJIBZ
64	RETURN ON INDICATOR BIT PAIR ONES	RIBO
65	DELAYED RETURN ON INDICATOR BIT PAIR ONES	DRIBO
66	RETURN ON INDICATOR BIT PAIR ZEROS	RIBZ
67	DELAYED RETURN ON INDICATOR BIT PAIR ZEROS	DRIBZ



TABLE 6-5—JUMP INSTRUCTIONS (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
68	JUMP ON FIELD ARRAY	JFA
69	JUMP ON FIELD ARRAY LINK	JFAL
6A	JUMP ON INDICATOR BIT PAIR ONES	JIBO
6B	JUMP ON INDICATOR BIT PAIR ZEROS	JIBZ
6C	JUMP RELATIVE PLUS	JRP
6D	JUMP RELATIVE MINUS	JRM
6E	DELAYED JUMP RELATIVE PLUS	DJRP
6F	DELAYED JUMP RELATIVE MINUS	DJRM
70-7F	JUMP ON INDICATOR ONES LINKAGE	JIOL
80-8F	JUMP ON INDICATOR ZEROS LINKAGE	JIZL
90	DELAYED JUMP ON TALLY NOT ZERO	DJTNZ
B0	JUMP RELATIVE PLUS EXTERNAL	JRPX
B1	JUMP RELATIVE MINUS EXTERNAL	JRMX
B2	JUMP ON INDICATOR BIT PAIR ONES MINUS	JIBOM
B3	JUMP ON INDICATOR BIT PAIR ZEROS MINUS	JIBZM
B4	DELAYED JUMP ON INDICATOR BIT PAIR ONES MINUS	DJIBOM
B5	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS MINUS	DJIBZM
B6	JUMP ON INDICATOR BIT PAIR ONES LONG	JIBOL
B7	JUMP ON INDICATOR BIT PAIR ZEROS LONG	JIBZL
B8	SKIP ON INDICATOR BIT PAIR ONES	SIBO
B9	SKIP ON INDICATOR BIT PAIR ZEROS	SIBZ
BA	JUMP ON TALLY NOT ZERO	JTNZ
C2	SKIP ON REGISTER UNEQUAL	SRU
C3	SKIP ON REGISTER EQUAL	SRE
C4	SKIP ON REGISTER BIT PAIR ONES	SRBO
C5	SKIP ON REGISTER BIT PAIR ZEROS	SRBZ
E0-EF	JUMP ON REGISTER ONES	JRO
F0-FF	JUMP ON REGISTER ZEROS	JRZ

TABLE 6-6—MISCELLANEOUS INSTRUCTIONS

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
12	EXTENDED ARITHMETIC FUNCTION	EAF
13	WAIT ON PMBUS	WPMB
2E	IBM SETUP ASSIST A	SETIA
2F	NVM SETUP ASSIST A	SETNA
3D	RECEIVE FETCHED DATA	RCV
54	MAP IBM INDICATORS	MII

TABLE 6-6—MISCELLANEOUS INSTRUCTIONS (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
55	MAP NVM INDICATORS	MIN
56	MAP VRX INDICATORS	MIV
61	LOAD LINK ADDRESS MINUS	LINKM
91	RETURN FROM TRAPS/INTERRUPTS	RTI
92	LOAD LINK ADDRESS	LINK
94	SET CONTROLS	SC
95	RESET CONTROLS	RC
96	TRANSFER BYTE FROM SETUP	TSB
97	TRANSFER LEFT DIGIT FROM SETUP AND CLEAR	TSLDC
98	TRANSFER RIGHT DIGIT FROM SETUP AND CLEAR	TSRDC
99	TRANSFER BYTE FROM SETUP AND CLEAR	TSBC
9A	SETUP SIGN EXTENSION	SETXS
9B	LOAD TALLY FROM SETUP	LTS
9E	SET CARRY TO ONE	SCO
9F	RESET INDICATORS TO ZERO	RIZ
D6	UNPACK LEFT DIGIT	UPKL
D7	UNPACK RIGHT DIGIT	UPKR

## INSTRUCTION INDEX BY OP CODE

Table 6-7 lists the CPC instruction set in ascending Op Code order.

TABLE 6-7—INSTRUCTION INDEX BY OP CODE

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
00-01	TRANSFER IN EXTERNAL (32-63)	TIE
02	MEMORY REFERENCE RETRY	MRR
03	FETCH REAL	FR
04	FETCH	F
05	LOAD, FETCH AND AUGMENT	LFA
06	LOAD, FETCH AND AUGMENT LINKAGE	LFAL
07	LOAD, FETCH AND DECREMENT	LFD
08-0B	FETCH LITERAL	FL
0C-0F	TRANSFER IN PORT (64-127)	TIP
10, 11	TRANSFER OUT EXTERNAL (32-63)	TOE
12	EXTENDED ARITHMETIC FUNCTION	EAF
13	WAIT ON PM BUS	WPMB

TABLE 6-7—INSTRUCTION INDEX BY OP CODE (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
14	STORE REAL	SR
15	STORE	S
16	STORE AND AUGMENT	SA
17	STORE AND DECREMENT	SD
18-1B	STORE LITERAL	SL
1C-1F	TRANSFER OUT PORT (64-127)	TOP
20, 21	TRANSFER IN INTERNAL (0-31)	TII
22	TRANSFER FIELD TO LEFT HALFWORD DECREMENT	TFLHD
23	TRANSFER FIELD TO LEFT HALFWORD INCREMENT	TFLHI
24	TRANSFER FIELD TO RIGHT HALFWORD DECREMENT	TFRHD
25	TRANSFER FIELD TO RIGHT HALFWORD INCREMENT	TFRHI
26	TRANSFER FIELD TO FIELD DECREMENT	TFFD
27	TRANSFER FIELD TO FIELD INCREMENT	TFFI
28	TRANSFER FIELD TO BYTE DECREMENT	TBBD
29	TRANSFER FIELD TO BYTE INCREMENT	TBBI
2A	TRANSFER FIELD TO BYTE DECREMENT NO TALLY	TBBDN
2B	TRANSFER FIELD TO BYTE	TFB
2C	JUMP ON REGISTER	JOR
2D	DELAYED JUMP ON REGISTER	DJOR
2E	IBM SETUP ASSIST A	SETIA
2F	NVM SETUP ASSIST A	SETNA
30, 31	TRANSFER OUT INTERNAL (0-31)	TOI
32	IBM SETUP JUMP A	JMPIA
33	IBM SETUP JUMP B	JMPIB
34	IBM SETUP JUMP C	JMPIC
35	NVM SETUP JUMP A	JMPNA
36	NVM SETUP JUMP B	JMPNB
37	NVM SETUP JUMP C	JMPNC
38	VRX SETUP JUMP A	JMPVA
39	VRX SETUP JUMP B	JMPVB
3A	VRX SETUP JUMP C	JMPVC
3B	NVM DESCRIPTOR JUMP	JMPD
3C	JUMP ON PMBUS NEGATIVE	JPMBN
3D	RECEIVE FETCHED DATA	RCV
3E	DELAYED JUMP ON PMBUS NEGATIVE	DJPMBN
3F	TRANSFER FIELD TO BYTE INCREMENT NO TALLY	TBBIN
40	ADD FIELD	AF

TABLE 6-7—INSTRUCTION INDEX BY OP CODE (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
41	SUBTRACT FIELD	SF
42	BOOLEAN AND FIELD	BAF
43	BOOLEAN OR FIELD	BOF
44	BOOLEAN EOR FIELD	BEF
45	COMPARE FIELD UNSIGNED	CFU
46	ADD PACKED DECIMAL FIELD	APDF
47	SUBTRACT PACKED DECIMAL FIELD	SPDF
48	ADD UNPACKED DECIMAL FIELD	AUDF
49	SUBTRACT UNPACKED DECIMAL FIELD	SUDF
4A	ADD WORD	AW
4B	SUBTRACT WORD	SW
4C	ADD WORD WITH CARRY	AWC
4D	SUBTRACT WORD WITH CARRY	SWC
4E	BOOLEAN AND WORD	BAW
4F	BOOLEAN OR WORD	BOW
50	BOOLEAN EOR WORD	BEW
51	TRANSFER WORD	TW
52	COMPARE WORD SIGNED	CWS
53	COMPARE WORD UNSIGNED	CWU
54	MAP IBM INDICATORS	MII
55	MAP NVM INDICATORS	MIN
56	MAP VRX INDICATORS	MIV
57	COMPARE BYTE TO FIELD UNSIGNED	CBFU
58	LOAD TALLY RIGHT CLEAR LEFT	LTRC
59	LOAD BYTE	LB
5A	LOAD LEFT DIGIT	LLD
5B	LOAD RIGHT DIGIT	LRD
5C	LOAD RIGHT HALFWORD	LRH
5D	LOAD RIGHT HALFWORD CLEAR LEFT	LRHC
5E	BOOLEAN INVERT WORD	BIW
60	SKIP ON REGISTER BYTE 3 ONES	SRB3O
61	LOAD LINK ADDRESS MINUS	LINKM
62	DELAYED JUMP ON INDICATOR BIT PAIR ONES	DJIBO
63	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS	DJIBZ
64	RETURN ON INDICATOR BIT PAIR ONES	RIBO
65	DELAYED RETURN ON INDICATOR BIT PAIR ONES	DRIBO
66	RETURN ON INDICATOR BIT PAIR ZEROES	RIBZ

TABLE 6-7—INSTRUCTION INDEX BY OP CODE (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
67	DELAYED RETURN ON INDICATOR BIT PAIR ZEROS	DRIBZ
68	JUMP ON FIELD ARRAY	JFA
69	JUMP ON FIELD ARRAY LINKAGE	JFAL
6A	JUMP ON INDICATOR BIT PAIR ONES	JIBO
6B	JUMP ON INDICATOR BIT PAIR ZEROS	JIBZ
6C	JUMP RELATIVE PLUS	JRP
6D	JUMP RELATIVE MINUS	JRM
6E	DELAYED JUMP RELATIVE PLUS	DJRP
6F	DELAYED JUMP RELATIVE MINUS	DJRM
70-7F	JUMP ON INDICATOR ONES LINKAGE	JIOL
80-8F	JUMP ON INDICATOR ZEROS LINKAGE	JIZL
90	DELAYED JUMP ON TALLY NOT ZERO	DJTNZ
91	RESTORE FROM TRAPS/INTERRUPTS	RTI
92	LOAD LINK ADDRESS	LINK
93	ADD CR TO LITERAL	ACRL
94	SET CONTROLS	SC
95	RESET CONTROLS	RC
96	TRANSFER BYTE FROM SETUP	TSB
97	TRANSFER LEFT DIGIT FROM SETUP AND CLEAR	TSLDC
98	TRANSFER RIGHT DIGIT FROM SETUP AND CLEAR	TSRDC
99	TRANSFER BYTE FROM SETUP AND CLEAR	TSBC
9A	SETUP SIGN EXTENSION	SETSX
9B	LOAD TALLY FROM SETUP	LTS
9C	ADD WORD WITH LITERAL NO INDICATOR CHANGE	AWLNI
9D	SUBTRACT WORD LITERAL NO INDICATOR CHANGE	SWLNI
9E	SET CARRY TO ONE	SCO
9F	RESET INDICATORS TO ZERO	RIZ
A0	SHIFT WORD ARITHMETIC RIGHT	SWAR
A1	SHIFT WORD LOGICAL LEFT	SWLL
A2	SHIFT WORD LOGICAL LEFT WITH CARRY	SWLLC
A3	SHIFT WORD LOGICAL RIGHT	SWLR
A4	SHIFT WORD LOGICAL RIGHT WITH CARRY	SWLRC
A5	SHIFT WORD CIRCULAR LEFT	SWCL
A6	TRANSFER LEFT HALFWORD TO FIELD DECREMENT	TLHFD
A7	TRANSFER LEFT HALFWORD TO FIELD INCREMENT	TLHFI
A8	TRANSFER RIGHT HALFWORD TO FIELD DECREMENT	TRHFD

TABLE 6-7—INSTRUCTION INDEX BY OP CODE (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
A9	TRANSFER RIGHT HALFWORD TO FIELD INCREMENT	TRHFI
AA	TRANSFER LEFT HALFWORD TO LEFT HALFWORD	TLHLH
AB	TRANSFER LEFT HALFWORD TO RIGHT HALFWORD	TLHRH
AC	TRANSFER RIGHT HALFWORD TO LEFT HALFWORD	TRHLH
AD	TRANSFER RIGHT HALFWORD TO RIGHT HALFWORD	TRHRH
AE	ADD WORD WITH LITERAL	AWL
AF	SUBTRACT WORD WITH LITERAL	SWL
B0	JUMP RELATIVE PLUS EXTERNAL	JRPX
B1	JUMP RELATIVE MINUS EXTERNAL	JRMX
B2	JUMP ON INDICATOR BIT PAIR ONES MINUS	JIBOM
B3	JUMP ON INDICATOR BIT PAIR ZEROS MINUS	JIBZM
B4	DELAYED JUMP ON INDICATOR BIT PAIR ONES MINUS	DJIBOM
B5	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS MINUS	DJIBZM
B6	JUMP ON INDICATOR BIT PAIR ONES LONG	JIBOL
B7	JUMP ON INDICATOR BIT PAIR ZEROS LONG	JIBZL
B8	SKIP ON INDICATOR BIT PAIR ONES	SIBO
B9	SKIP ON INDICATOR BIT PAIR ZEROS	SIBZ
BA	JUMP ON TALLY NOT ZERO	JTNZ
BB	TRANSFER BYTE TO FIELD INCREMENT NO TALLY	TBFIN
BC	TRANSFER BYTE TO FIELD DECREMENT	TBFD
BD	TRANSFER BYTE TO FIELD INCREMENT	TBFI
BE	TRANSFER BYTE TO FIELD DECREMENT NO TALLY	TBFDN
BF	TRANSFER BYTE TO FIELD	TBF
C0	COMPARE BYTE SIGNED	CBS
C1	COMPARE BYTE UNSIGNED	CBU
C2	SKIP ON REGISTER UNEQUAL	SRU
C3	SKIP ON REGISTER EQUAL	SRE
C4	SKIP ON REGISTER BIT PAIR ONES	SRBO
C5	SKIP ON REGISTER BIT PAIR ZEROS	SRBZ
C6	ADD BYTE	AB
C7	SUBTRACT BYTE	SB
C8	ADD BYTE WITH CARRY	ABC
C9	SUBTRACT BYTE WITH CARRY	SBC
CA	BOOLEAN AND BYTE	BAB
CB	BOOLEAN OR BYTE	BOB

TABLE 6-7—INSTRUCTION INDEX BY OP CODE (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
CC	BOOLEAN EOR BYTE	BEB
CD	TRANSFER BYTE	TB
CE	ADD PACKED DECIMAL BYTE	APDB
CF	SUBTRACT PACKED DECIMAL BYTE	SPDB
D0	ADD PACKED DECIMAL BYTE WITH CARRY	APDBC
D1	SUBTRACT PACKED DECIMAL BYTE WITH CARRY	SPDBC
D2	TRANSFER LEFT DIGIT TO LEFT DIGIT	TLDL
D3	TRANSFER LEFT DIGIT TO RIGHT DIGIT	TLDR
D4	TRANSFER RIGHT DIGIT TO LEFT DIGIT	TRDL
D5	TRANSFER RIGHT DIGIT TO RIGHT DIGIT	TRDR
D6	UNPACK LEFT DIGIT	UPKL
D7	UNPACK RIGHT DIGIT	UPKR
D8	BOOLEAN AND LEFT DIGIT	BALD
D9	BOOLEAN AND RIGHT DIGIT	BARD
DA	BOOLEAN OR LEFT DIGIT	BOLD
DB	BOOLEAN OR RIGHT DIGIT	BORD
DC	BOOLEAN EOR LEFT DIGIT	BELD
DD	BOOLEAN EOR RIGHT DIGIT	BERD
DE	SUBTRACT BYTE LITERAL	SBL
DF	BOOLEAN INVERT BYTE	BIB
E0-EF	JUMP ON REGISTER ONES	JRO
F0-FF	JUMP ON REGISTER ZEROS	JRZ

## INSTRUCTION INDEX BY MNEMONIC

Table 6-8 lists the CPC instruction set by alphabetical order of mnemonic.

TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
C6	ADD BYTE	AB
C8	ADD BYTE WITH CARRY	ABC
93	ADD CR LITERAL	ACRL
40	ADD FIELD	AF
CE	ADD PACKED DECIMAL BYTE	APDB
DO	ADD PACKED DECIMAL BYTE WITH CARRY	APDBC
46	ADD PACKED DECIMAL FIELD	APDF
48	ADD UNPACKED DECIMAL FIELD	AUDF

TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
4A	ADD WORD	AW
4C	ADD WORD WITH CARRY	AWC
AE	ADD WORD WITH LITERAL	AWL
9C	ADD WORD WITH LITERAL NO INDICATOR CHANGE	AWLNI
CA	BOOLEAN AND BYTE	BAB
42	BOOLEAN AND FIELD	BAF
D8	BOOLEAN AND LEFT DIGIT	BALD
D9	BOOLEAN AND RIGHT DIGIT	BARD
4E	BOOLEAN AND WORD	BAW
CC	BOOLEAN EOR BYTE	BEB
44	BOOLEAN EOR FIELD	BEF
DC	BOOLEAN EOR LEFT DIGIT	BELD
DD	BOOLEAN EOR RIGHT DIGIT	BERD
50	BOOLEAN EOR WORD	BEW
DF	BOOLEAN INVERT BYTE	BIB
5E	BOOLEAN INVERT WORD	BIW
CB	BOOLEAN OR BYTE	BOB
43	BOOLEAN OR FIELD	BOF
DA	BOOLEAN OR LEFT DIGIT	BOLD
DB	BOOLEAN OR RIGHT DIGIT	BORD
4F	BOOLEAN OR WORD	BOW
C0	COMPARE BYTE SIGNED	CBS
C1	COMPARE BYTE UNSIGNED	CBU
57	COMPARE BYTE TO FIELD UNSIGNED	CBFU
45	COMPARE FIELD UNSIGNED	CFU
52	COMPARE WORD SIGNED	CWS
53	COMPARE WORD UNSIGNED	CWU
62	DELAYED JUMP ON INDICATOR BIT PAIR ONES	DJIBO
B4	DELAYED JUMP ON INDICATOR BIT BIT PAIR ONES MINUS	DJIBOM
63	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS	DJIBZ
B5	DELAYED JUMP ON INDICATOR BIT PAIR ZEROS MINUS	DJIBZM
2D	DELAYED JUMP ON REGISTER	DJOR
3E	DELAYED JUMP ON PMBUS NEGATIVE	DJPMBN
6F	DELAYED JUMP RELATIVE MINUS	DJRM
6E	DELAYED JUMP RELATIVE PLUS	DJRP
90	DELAYED JUMP ON TALLY NOT ZERO	DJTNZ
65	DELAYED RETURN ON INDICATOR BIT PAIR ONES	DRIBO
67	DELAYED RETURN ON INDICATOR BIT PAIR ZEROS	DRIBZ
12	EXTENDED ARITHMETIC FUNCTION	EAF



TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
04	FETCH	F
08-0B	FETCH LITERAL	FL
03	FETCH REAL	FR
68	JUMP ON FIELD ARRAY	JFA
69	JUMP ON FIELD ARRAY LINK	JFAL
6A	JUMP ON INDICATOR BIT PAIR ONES	JIBO
B6	JUMP ON INDICATOR BIT PAIR ONES LONG	JIBOL
B2	JUMP ON INDICATOR BIT PAIR ONES MINUS	JIBOM
6B	JUMP ON INDICATOR BIT PAIR ZEROS	JIBZ
B7	JUMP ON INDICATOR BIT PAIR ZEROS LONG	JIBZL
B3	JUMP ON INDICATOR BIT PAIR ZEROS MINUS	JIBZM
70-7F	JUMP ON INDICATOR ONES LINKAGE	JIOL
80-8F	JUMP ON INDICATOR ZEROS LINKAGE	JIZL
3B	NVM DESCRIPTOR JUMP	JMPD
32	IBM SETUP JUMP A	JMPIA
33	IBM SETUP JUMP B	JMPIB
34	IBM SETUP JUMP C	JMPIC
35	NVM SETUP JUMP A	JMPNA
36	NVM SETUP JUMP B	JMPNB
37	NVM SETUP JUMP C	JMPNC
38	VRX SETUP JUMP A	JMPVA
39	VRX SETUP JUMP B	JMPVB
3A	VRX SETUP JUMP C	JMPVC
2C	JUMP ON REGISTER	JOR
3C	JUMP ON PM BUS NEGATIVE	JPMBN
6D	JUMP RELATIVE MINUS	JRM
B1	JUMP RELATIVE MINUS EXTERNAL	JRMX
E0-EF	JUMP ON REGISTER ONES	JRO
6C	JUMP RELATIVE PLUS	JRP
B0	JUMP RELATIVE PLUS EXTERNAL	JRPX
F0-FF	JUMP ON REGISTER ZEROS	JRZ
BA	JUMP ON TALLY NOT ZERO	JTNZ
59	LOAD BYTE	LB
05	LOAD, FETCH, AND AUGMENT	LFA
06	LOAD, FETCH, AND AUGMENT LINKAGE	LFAL
07	LOAD, FETCH, AND DECREMENT	LFD
92	LOAD LINK ADDRESS	LINK
61	LOAD LINK ADDRESS MINUS	LINKM
5A	LOAD LEFT DIGIT	LLD
5B	LOAD RIGHT DIGIT	LRD
5C	LOAD RIGHT HALFWORD	LRH
5D	LOAD RIGHT HALFWORD CLEAR LEFT HALFWORD	LRHC

TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
58	LOAD TALLY RIGHT CLEAR LEFT	LTRC
9B	LOAD TALLY FROM SETUP	LTS
54	MAP IBM INDICATORS	MII
55	MAP NVM INDICATORS	MIN
56	MAP VRX INDICATORS	MIV
02	MEMORY REFERENCE RETRY	MRR
95	RESET CONTROLS	RC
3D	RECEIVE FETCHED DATA	RCV
64	RETURN ON INDICATOR BIT PAIR ONES	RIBO
66	RETURN ON INDICATOR BIT PAIR ZEROS	RIBZ
9F	RESET INDICATORS TO ZERO	RIZ
91	RETURN FROM TRAPS/INTERRUPTS	RTI
15	STORE	S
16	STORE AND AUGMENT	SA
C7	SUBTRACT BYTE	SB
C9	SUBTRACT BYTE WITH CARRY	SBC
DE	SUBTRACT BYTE LITERAL	SBL
94	SET CONTROLS	SC
9E	SET CARRY TO ONE	SCO
17	STORE AND DECREMENT	SD
2E	IBM SETUP ASSIST A	SETIA
2F	NVM SETUP ASSIST A	SETNA
9A	SETUP SIGN EXTENSION	SETSX
41	SUBTRACT FIELD	SF
B8	SKIP ON INDICATOR BIT PAIR ONES	SIBO
B9	SKIP ON INDICATOR BIT PAIR ZEROS	SIBZ
18-1B	STORE LITERAL	SL
CF	SUBTRACT PACKED DECIMAL BYTE	SPDB
D1	SUBTRACT PACKED DECIMAL BYTE WITH CARRY	SPDBC
47	SUBTRACT PACKED DECIMAL FIELD	SPDF
14	STORE REAL	SR
C4	SKIP ON REGISTER BIT PAIR ONES	SRBO
C5	SKIP ON REGISTER BIT PAIR ZEROS	SRBZ
60	SKIP ON REGISTER BYTE 3 ONES	SRB3O
C3	SKIP ON REGISTERS EQUAL	SRE
C2	SKIP ON REGISTERS UNEQUAL	SRU
49	SUBTRACT UNPACKED DECIMAL FIELD	SUDF
4B	SUBTRACT WORD	SW
A0	SHIFT WORD ARITHMETIC RIGHT	SWAR
4D	SUBTRACT WORD WITH CARRY	SWC
A5	SHIFT WORD CIRCULAR LEFT	SWCL

TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
AF	SUBTRACT WORD WITH LITERAL	SWL
A1	SHIFT WORD LOGICAL LEFT	SWLL
A2	SHIFT WORD LOGICAL LEFT WITH CARRY	SWLLC
9D	SUBTRACT WORD LITERAL NO INDICATOR CHANGE	SWLNI
A3	SHIFT WORD LOGICAL RIGHT	SWLR
A4	SHIFT WORD LOGICAL RIGHT WITH CARRY	SWLRC
CD	TRANSFER BYTE	TB
BF	TRANSFER BYTE TO FIELD	TBF
BC	TRANSFER BYTE TO FIELD DECREMENT	TBFD
BE	TRANSFER BYTE TO FIELD DECREMENT NO TALLY CHANGE	TBFDN
BD	TRANSFER BYTE TO FIELD INCREMENT	TBFI
BB	TRANSFER BYTE TO FIELD INCREMENT NO TALLY CHANGE	TBFIN
2B	TRANSFER FIELD TO BYTE	TFB
28	TRANSFER FIELD TO BYTE DECREMENT	TFBD
2A	TRANSFER FIELD TO BYTE DECREMENT NO TALLY CHANGE	TFBDN
29	TRANSFER FIELD TO BYTE INCREMENT	TFBI
3F	TRANSFER FIELD TO BYTE INCREMENT NO TALLY CHANGE	TFBIN
26	TRANSFER FIELD TO FIELD DECREMENT	TFFD
27	TRANSFER FIELD TO FIELD INCREMENT	TFFI
22	TRANSFER FIELD TO LEFT HALFWORD DECREMENT	TFLHD
23	TRANSFER FIELD TO LEFT HALFWORD INCREMENT	TFLHI
24	TRANSFER FIELD TO RIGHT HALFWORD DECREMENT	TFRHD
25	TRANSFER FIELD TO RIGHT HALFWORD INCREMENT	TFRHI
00-01	TRANSFER IN EXTERNAL (32-63)	TIE
20-21	TRANSFER IN INTERNAL (0-31)	TII
0C-0F	TRANSFER IN PORT (64-127)	TIP
D2	TRANSFER LEFT DIGIT TO LEFT DIGIT	TLDL
D3	TRANSFER LEFT DIGIT TO RIGHT DIGIT	TLDRD
A6	TRANSFER LEFT HALFWORD TO FIELD DECREMENT	TLHFD
A7	TRANSFER LEFT HALFWORD TO FIELD INCREMENT	TLHFI
AA	TRANSFER LEFT HALFWORD TO LEFT HALFWORD	TLHLH
AB	TRANSFER LEFT HALFWORD TO RIGHT HALFWORD	TLHRH
10, 11	TRANSFER OUT EXTERNAL (32-63)	TOE

TABLE 6-8—INSTRUCTION INDEX BY MNEMONIC (Continued)

OP CODE HEX	INSTRUCTION NAME	MNEMONIC
30, 31	TRANSFER OUT INTERNAL (0-31)	TOI
1C-1F	TRANSFER OUT PORT (64-127)	TOP
D4	TRANSFER RIGHT DIGIT TO LEFT DIGIT	TRDL
D5	TRANSFER RIGHT DIGIT TO RIGHT DIGIT	TRDR
A8	TRANSFER RIGHT HALFWORD TO FIELD DECREMENT	TRHFD
A9	TRANSFER RIGHT HALFWORD TO FIELD INCREMENT	TRHFI
AC	TRANSFER RIGHT HALFWORD TO LEFT HALFWORD	TRHLH
AD	TRANSFER RIGHT HALFWORD TO RIGHT HALFWORD	TRHRH
96	TRANSFER BYTE FROM SETUP	TSB
99	TRANSFER BYTE FROM SETUP AND CLEAR	TSBC
97	TRANSFER LEFT DIGIT FROM SETUP AND CLEAR	TSLDC
98	TRANSFER RIGHT DIGIT FROM SETUP AND CLEAR	TSRDC
51	TRANSFER WORD	TW
D6	UNPACK LEFT DIGIT	UPKL
D7	UNPACK RIGHT DIGIT	UPKR
13	WAIT ON PMBUS	WPMB

The following pages contain descriptions of all CPC instructions. The op code, mnemonic, format, summary, operation description, number of cycles, indicator array effect, and relevant programming convention is given for each instruction.

**AB****ADD BYTE****AB****MNEMONIC:** AB**OP CODE:** C6**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	0	1	1	0	Source Dest RSU		Source Dest RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field			K Field				

GIM2216

**SUMMARY:**  $(RJ - B) + (RK - B) \rightarrow RJ - B$ 

**OPERATION:** A byte from the RSU specified by the J FIELD is binarily added to a byte from the RSU specified by the K FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** AB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, and I4.

**PROGRAMMING CONVENTIONS:** None

**ABC****ADD BYTE WITH CARRY****ABC****MNEMONIC:** ABC**OP CODE:** C8**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	1	0	0	0	Source Dest RSU	Source Dest RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2218

**SUMMARY:**  $(RJ - B) + (RK - B) + C \rightarrow RJ - B$ 

**OPERATION:** A byte from the RSU specified by the J FIELD is added binarily to a byte from the RSU specified by the K FIELD and the Carry Bit (I4) from a previous instruction. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** ABC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, and I4.

**PROGRAMMING CONVENTIONS:** None

**ACRL**

ADD CR LITERAL

**ACRL**

MNEMONIC: ACRL

OP CODE: 93

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	0	0	1	1	Dest RSU				Digit Literal			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2214

SUMMARY: Control Store Address +K  $\rightarrow$  RJ

**OPERATION:** The Control Store Address of the ACRL instruction is augmented by the digit literal in the K FIELD and transferred to the right halfword in the RSU specified by the J FIELD. The left halfword of the destination RSU is not affected.

**NUMBER OF CYCLES:** ACRL is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**AF****ADD FIELD****AF****MNEMONIC:** AF**OP CODE:** 40**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	0	0	0	0	0	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2219

**SUMMARY:**  $(RJ - B) + (RK - B) + C \rightarrow R13$ ; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers is binarily added to a byte from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the byte in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. Following the addition, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** AF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding AF must not alter the Tally Register or the MARS Address Registers used by AF.



**APDB****ADD PACKED DECIMAL BYTE****APDB****MNEMONIC:** APDB**OP CODE:** CE**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1 1 0 0 1 1 1 0								Source Dest RSU		Source Dest RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field				K Field			

GIM2217

**SUMMARY:**  $(RJ - B) + (RK - B) \rightarrow RJ - B$ 

**OPERATION:** A byte from the RSU specified by the J FIELD is added packed decimally to a byte from the RSU specified by the K FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** APDB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

# APDBC      ADD PACKED DECIMAL BYTE WITH CARRY      APDBC

**MNEMONIC:** APDBC

**OP CODE:** D0

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	0	0	0	0	Source Dest RSU		Source Dest RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field			K Field				

GIM2220

**SUMMARY:**  $(RJ - B) + (RK - B) + C \rightarrow (RJ - B)$

**OPERATION:** A byte from the RSU specified by the J FIELD is added packed decimally to a byte from the RSU specified by the K FIELD and the Carry Bit (I4) from a previous instruction. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** APDBC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

**APDF****ADD PACKED DECIMAL FIELD****APDF****MNEMONIC:** APDF**OP CODE:** 46**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	0	0	1	1	0	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2221

**SUMMARY:**  $(RJ - B) + (RK - B) + C \rightarrow R13$ ; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers is added packed decimally to a byte from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the byte in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. Following the addition, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** APDF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4 and I5.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding APDF must not alter the Tally Register or the MARS Address Registers used by APDF.

**AUDF**      ADD UNPACKED DECIMAL FIELD      **AUDF**

MNEMONIC:    AUDF

OP CODE:      48

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	1
	OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2222

**SUMMARY:**       $(RJ - B) + (RK - B) + C \rightarrow R13$ ; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, the low-order digit from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers is added decimally to the low-order digit from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the low-order digit in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. The Hex value 0011 is loaded into the high-order digit as the ASCII zone character. Following the addition, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** AUDF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I4 and I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding AUDF must not alter the Tally Register or the MARS Address Registers used by AUDF.

**AW**

**ADD WORD**

**AW**

**MNEMONIC:** AW

**OP CODE:** 4A

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	0	1	0	Source Dest RSU		Source RSU					
OP Code; G, H, & I Fields								J Field				K Field			

GIM2211

**SUMMARY:** (JR) + (RK) → RJ

**OPERATION:** A word from the RSU specified by the J FIELD is binarily added to a word from the RSU specified by the K FIELD. The result replaces the operand in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** AW is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

**AWC****ADD WORD WITH CARRY****AWC****MNEMONIC:** AWC**OP CODE:** 4C**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	0	1	1	0	0	Source Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2212

**SUMMARY:**  $(RJ) + (RK) + C \rightarrow RJ$ 

**OPERATION:** A word from the RSU specified by the J FIELD is binarily added to a word from the RSU specified by the K FIELD and the Carry Bit (I4) from a previous instruction. The result replaces the operand in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** AWC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

AWL

ADD WORD WITH LITERAL

AWL

MNEMONIC:   AWL

OP CODE:      AE

FORMAT:       16 15 14 13 12 11 10 9   8 7 6 5 4 3 2 1

1	0	1	0	1	1	1	0	Source Dest RSU				Digit Literal			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2213

SUMMARY:     (RJ) + K → RJ

OPERATION:   A word from the RSU specified by the J FIELD is augmented by the digit literal in the K FIELD. The result replaces the word in the RSU specified by the J FIELD.

NUMBER OF CYCLES:

EFFECT ON INDICATOR ARRAY:   This instruction affects Indicators I1, I2, I3, I4, and I5.

PROGRAMMING CONVENTIONS:   None

**AWLNI****ADD WORD WITH LITERAL  
NO INDICATOR CHANGE****AWLNI****MNEMONIC:** AWLNI**OP CODE:** 9C**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	1	1	0	0	Source Dest RSU				Digit Literal			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2215

**SUMMARY:**  $(RJ) + K \rightarrow RJ$ 

**OPERATION:** A word from the RSU specified by the J FIELD is augmented by the digit literal in the K FIELD. The result replaces the word in the RSU specified in the J FIELD.

**NUMBER OF CYCLES:** AWLNI is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None



**BAB****BOOLEAN AND BYTE****BAB****MNEMONIC:** BAB**OP CODE:** CA

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	1	0	0	1	0	1	0	Source Dest RSU	Source Dest Byte	Source RSU	Source RSU Byte				
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2199

**SUMMARY:** (RJ - B) AND (RK - B) → RJ - B

**OPERATION:** A byte from the RSU specified by the J FIELD is ANDed with a byte from the RSU specified by the K FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** BAB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

# BAF

## BOOLEAN AND FIELD

# BAF

**MNEMONIC:** BAF

**OP CODE:** 42

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	0	0	1	0	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2206

**SUMMARY:** (RJ - B) AND (RK - B) → R13; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from MARS4 Data Register (RSU9) specified by the J FIELD and the MARS4 Byte Pointers is logically ANDed with a byte from the MARS5 Data Register (RSU11) specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the MARS6 Data Register specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are decremented by one and if any of the Byte Pointers crossed the word boundary, then the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** BAF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding BAF must not alter the Tally Register or the MARS Address Registers used by BAF.

**BALD**

**BOOLEAN AND LEFT DIGIT**

**BALD**

**MNEMONIC:** BALD

**OP CODE:** D8

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	1	0	0	0	Source Dest RSU		Source Dest RSU Byte					Digital Literal
OP Code; G, H, & I Fields								J Field				K Field			

GIM2202

**SUMMARY:** (RJ - LD) AND K → RJ - LD

**OPERATION:** The left digit of a byte from the RSU specified by the J FIELD is ANDed with the digit literal in the K FIELD. The result replaces the left digit of the byte in the RSU specified by the J FIELD. The right digit remains unchanged.

**NUMBER OF CYCLES:** BALD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

**PROGRAMMING CONVENTIONS:** None

**BARD**

BOOLEAN AND RIGHT DIGIT

**BARD**

MNEMONIC: BARD

OP CODE: D9

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	1	1	0	0	1	Source Dest RSU	Source Dest RSU Byte	Digit Literal
OP Code; G, H, & I Fields								J Field		K Field

GIM2203

SUMMARY: (RJ - RD) AND K → RJ - RD

**OPERATION:** The right digit of a byte from the RSU specified by the J FIELD is ANDed with the digit literal in the K FIELD. The result replaces the right digit of the byte in the RSU specified by the J FIELD. The left digit remains unchanged.

**NUMBER OF CYCLES:** BARD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

**PROGRAMMING CONVENTIONS:** None

BAW

BOOLEAN AND WORD

BAW

MNEMONIC: BAW

OP CODE: 4E

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0 1 0 0 1 1 1 0	Source Dest RSU	Source RSU
OP Code; G, H, & I Fields	J Field	K Field

GIM2196

SUMMARY: (RJ) AND (RK) → RJ

OPERATION: A word from the RSU specified by the J FIELD is ANDed with a word from the RSU specified by the K FIELD. The result replaces the operand in the RSU specified by the J FIELD.

NUMBER OF CYCLES: BAW is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I1, I2, and I3.

PROGRAMMING CONVENTIONS: None

**BEB**

**BOOLEAN EOR BYTE**

**BEB**

**MNEMONIC:** BEB

**OP CODE:** CC

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	1	1	0	0	Source Dest RSU	Source Dest RSU Byte	Source RSU	Source RSU Byte				
OP Code; G, H, & I Fields								J Field		K Field					

GIM2201

**SUMMARY:** (RJ - B) EOR (RK - B) → RJ - B

**OPERATION:** A byte from the RSU specified by the J FIELD is EORed with a byte from the RSU specified by the K FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** BEB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

BEF

BOOLEAN EOR FIELD

BEF

MNEMONIC: BEF

OP CODE: 44

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	0	1	0	0	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2210

SUMMARY: (RJ - B) EOR (RK - B) → R13; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

OPERATION: If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from MARS 4 Data Register (RSU9) specified by the J FIELD and the MARS4 Byte Pointers is logically EORed with a byte from the MARS5 Data Register (RSU11) specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the MARS6 Data Register specified by the MARS6 Byte Pointers. Following the transfer the Byte Pointers are decremented by one and if any of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

NUMBER OF CYCLES: BEF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: The instruction immediately preceding BEF must not alter the Tally Register or the MARS Address Registers used by BEF.

**BELD****BOOLEAN EOR LEFT DIGIT****BELD****MNEMONIC:** BELD**OP CODE:** DC**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1 1 0 1 1 1 0 0								Source Dest RSU		Source Dest RSU Byte		Digit Literal			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2208

**SUMMARY:** (RJ - LD) EOR K → RJ - LD

**OPERATION:** The left digit of a byte from the RSU specified by the J FIELD is EORed with the digit literal in the K FIELD. The result replaces the left digit of the byte in the RSU specified by the J FIELD. The right digit remains unchanged.

**NUMBER OF CYCLES:** BELD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

**PROGRAMMING CONVENTIONS:** None



**BERD****BOOLEAN EOR RIGHT DIGIT****BERD****MNEMONIC:** BERD**OP CODE:** DD**FORMAT:**

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	1	1	1	0	1	Source Dest RSU	Source Dest RSU Byte	Digit Literal
OP Code; G, H, & I Fields								J Field		K Field

GIM2207

**SUMMARY:** (RJ - RD) EOR K → RJ - RD

**OPERATION:** The right digit of a byte from the RSU specified by the J FIELD is EORed with the digit literal in the K FIELD. The result replaces the right digit of the byte in the RSU specified by the J FIELD. The left digit remains unchanged.

**NUMBER OF CYCLES:** BERD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

**PROGRAMMING CONVENTIONS:** None

**BEW****BOOLEAN EOR WORD****BEW****MNEMONIC:** BEW**OP CODE:** 50**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	0	0	0	Source Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2198

**SUMMARY:** (RJ) EOR (RK) → RJ

**OPERATION:** A word from the RSU specified by the J FIELD is EORed with a word from the RSU specified by the K FIELD. The result replaces the operand in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** BEW is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

**BIB**

**BOOLEAN INVERT BYTE**

**BIB**

**MNEMONIC:** BIB

**OP CODE:** DF

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	1	1	1	1	Dest RSU		Dest RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field			K Field				

GIM2314

**SUMMARY:** (RK - B)/ ➡ RJ - B

**OPERATION:** A byte from the RSU specified by the K FIELD is ones complemented. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** BIB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

**BIW****BOOLEAN INVERT WORD****BIW****MNEMONIC:** BIW**OP CODE:** 5E**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	1	1	1	0	Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2298

**SUMMARY:** (RK)/ → RJ

**OPERATION:** A word from the RSU specified by the K FIELD is ones complemented. The result replaces the operand in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** BIW is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

BOB

BOOLEAN OR BYTE

BOB

MNEMONIC: BOB

OP CODE: CB

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	1	0	1	1	Source Dest RSU		Source Dest RSU Byte		Source RSU		Source RSU Byte	
Op Code; G, H, & I Fields								J Field			K Field				

GIM2200

SUMMARY: (RJ - B) OR (RK - B) → RJ - B

OPERATION: A byte from the RSU specified by the J FIELD is ORed with a byte from the RSU specified by the K FIELD. The result replaces the byte in the RSU specified by the J FIELD.

NUMBER OF CYCLES: BOB is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I1, I2, and I3.

PROGRAMMING CONVENTIONS: None

**BOF****BOOLEAN OR FIELD****BOF****MNEMONIC:** BOF**OP CODE:** 43**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	0	0	0	1	1	1	0	0	1	1	0	1	1
DP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2209

**SUMMARY:** (RJ - B) OR (RK - B) → R13; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from MARS4 Data Register (RSU9) specified by the J FIELD and the MARS4 Byte Pointers is logically ORed with a byte from the MARS5 Data Register (RSU11) specified by the K FIELD and the MARS5 Byte Pointers. The result is placed in the MARS6 Data Register specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are decremented by one and if any of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** BOF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding BOF must not alter the Tally Register or the MARS Address Registers used by BOF.

**BOLD**

BOOLEAN OR LEFT DIGIT

**BOLD**

MNEMONIC: BOLD

OP CODE: DA

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	1	0	1	0	Source Dest RSU		Source Dest RSU Byte			Digit Literal		
OP Code; G, H, & I Fields								J Field				K Field			

GIM2204

SUMMARY: (RJ - LD) OR K → RJ - LD

OPERATION: The left digit of a byte from the RSU specified by the J FIELD is ORed with the digit literal in the K FIELD. The result replaces the left digit of the byte in the RSU specified by the J FIELD. The right digit remains unchanged.

NUMBER OF CYCLES: BOLD is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

PROGRAMMING CONVENTIONS: None

**BORD****BOOLEAN OR RIGHT DIGIT****BORD****MNEMONIC:** BORD**OP CODE:** DB**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1 1 0 1 1 0 1 1								Source Dest RSU		Source Dest RSU Byte		Digit Literal			
OP Code; G, H, & I Field								J Field				K Field			

GIM2205

**SUMMARY:** (RJ - RD) OR K → RJ - RD

**OPERATION:** The right digit of a byte from the RSU specified by the J FIELD is ORed with the digit literal in the K FIELD. The result replaces the right digit of the byte in the RSU specified by the J FIELD. The left digit remains unchanged.

**NUMBER OF CYCLES:** BORD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3. However, only I2 is valid.

**PROGRAMMING CONVENTIONS:** None



**BOW**

BOOLEAN OR WORD

**BOW**

MNEMONIC: BOW

OP CODE: 4F

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	1	1	1	Source Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2197

SUMMARY: (RJ) OR (RK) → RJ

OPERATION: A word from the RSU specified by the J FIELD is ORed with a word from the RSU specified by the K FIELD. The result replaces the operand in the RSU specfied by the J FIELD.

NUMBER OF CYCLES: BOW is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I1, I2, and I3.

PROGRAMMING CONVENTIONS: None

**CBS****COMPARE BYTE SIGNED****CBS****MNEMONIC:** CBS**OP CODE:** CO**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	0	0	0	0	Source RSU	Source RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2237

**SUMMARY:** (RJ - B) > (RK - B) → I1-0, I2-0, I3-1  
 (RJ - B) < (RK - B) → I1-1, I2-0, I3-0  
 (RJ - B) = (RK - B) → I1-0, I2-1, I3-0

**OPERATION:** The signed contents of the byte of the RSU specified by the J FIELD are compared with the signed contents of the byte of the RSU specified by the K FIELD.

Indicators I1-I3 are set to indicate the byte in RSU-J is less than, greater than, or equal to the byte in RSU-K.

I3 - Greater  
 I2 - Equal  
 I1 - Less

**NUMBER OF CYCLES:** CBS is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

**CBU****COMPARE BYTE UNSIGNED****CBU****MNEMONIC:** CBU**OP CODE:** C1**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	0	0	0	1	Source RSU	Source RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2238

**SUMMARY:**      $(RJ - B) > (RK - B) \rightarrow I1-0, I2-0, I3-1$   
                       $(RJ - B) < (RK - B) \rightarrow I1-1, I2-0, I3-0$   
                       $(RJ - B) = (RK - B) \rightarrow I1-0, I2-1, I3-0$

**OPERATION:** The unsigned contents of the byte of the RSU specified by the J FIELD are compared with the unsigned contents of the byte of the RSU specified by the K FIELD.

Indicators I1-I3 are set to indicate the byte in RSU-J is less than, greater than, or equal to the byte in RSU-K.

I3 - Greater

I2 - Equal

I1 - Less

**NUMBER OF CYCLES:** CBU is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

**CBFU COMPARE BYTE TO FIELD UNSIGNED CBFU****MNEMONIC:** CBFU**OP CODE:** 57

<b>FORMAT:</b>															
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	0	1	1	1	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2240

**SUMMARY:** (RJ - B) - (RK - B); set Indicator I1, I2, or I3; increment RK Byte Pointers; decrement Tally Register; set M5OF if word boundary

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS4 Data Register (RSU9) specified by the J FIELD and the MARS4 Byte Pointers is compared with the byte from the MARS5 Data Register (RSU11) specified by the K FIELD and the MARS5 Byte Pointers. Indicator I1 is set if the MARS4 data is less than, I2 is set if the MARS4 data is equal to and I3 is set if the MARS4 data is greater than the MARS5 data. Indicators I5 and I6 are set by the BCD checks.

Following the compare, the MARS5 Byte Pointers are incremented by one, and if they crossed the word boundary, MARS5 Overflow Flag (M5OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a zero.

**NUMBER OF CYCLES:** CBFU is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS5 overflow occurs, until Indicator I1 or I3 is set, or until the Tally Register equals zero. If the exit condition is due to the setting of Indicator I1 or I3, no overflow will be detected.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, I5 and I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding CBFU must not alter the Tally Register or the MARS Address Registers used by the CBFU.

# CFU

## COMPARE FIELD UNSIGNED

# CFU

MNEMONIC: CFU

OP CODE: 45

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	0	1	0	1	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2239

**SUMMARY:** (RJ - B) - (RK - B); set Indicator I1, I2, or I3; increment Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS4 Data Register (RSU9) specified by the J FIELD and the MARS4 Byte Pointers is compared with the byte from the MARS5 Data Register (RSU11) specified by the K FIELD and the MARS5 Byte Pointers. Indicator I1 is set if the MARS4 data is less than, I2 is set if the MARS4 data is equal to and I3 is set if the MARS4 data is greater than the MARS5 data. Indicator I5 and I6 are set by the BCD checks.

Following the compare, the Byte Pointers are incremented by one, and if any of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a zero.

**NUMBER OF CYCLES:** CFU is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs, until I1 or I3 is set, or until the Tally Register equals zero. If I1 or I3 is set at the same time a MARS overflow occurs, the overflow is ignored.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, I5, and I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding CFU must not alter the Tally Register or the MARS Address Registers used by CFU.

**CWS****COMPARE WORD SIGNED****CWS****MNEMONIC:** CWS**OP CODE:** 52**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	0	1	0	Source RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2235

**SUMMARY:** (RJ) > (RK) → I1-0, I2-0, I3-1

(RJ) &lt; (RK) → I1-1, I2-0, I3-0

(RJ) = (RK) → I1-0, I2-1, I3-0

**OPERATION:** The signed contents of RSU (32 bits) specified by the J FIELD are compared with the signed contents of the RSU (32 bits) specified by the K FIELD.

Indicators I1-I3 are set to indicate the contents of RSU-J are less than, greater than, or equal to the contents of RSU-K.

I3 - Greater

I2 - Equal

I1 - Less

**NUMBER OF CYCLES:** CWS is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, and I3.

**PROGRAMMING CONVENTIONS:** None

CWU

COMPARE WORD UNSIGNED

CWU

MNEMONIC: CWU

OP CODE: 53

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	0	1	1	Source RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2236

SUMMARY: (RJ) > (RK) → I1-0, I2-0, I3-1  
(RJ) < (RK) → I1-1, I2-0, I3-0  
(RJ) = (RK) → I1-0, I2-1, I3-0

OPERATION: The unsigned contents of RSU (32 bits) specified by the J FIELD are compared with the unsigned contents of the RSU (32 bits) specified by the K FIELD.

Indicators I1-I3 are set to indicate the contents of RSU-J are less than, greater than, or equal to the contents of RSU-K.

- I3 - Greater
- I2 - Equal
- I1 - Less

NUMBER OF CYCLES: CWU is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I1, I2 and I3.

PROGRAMMING CONVENTIONS: None

**DJIBO****DELAYED JUMP ON  
INDICATOR BIT PAIR ONES****DJIBO****MNEMONIC:** DJIBO**OP CODE:** 62**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	1	0	0	0	1	0	Bit Pair Selctr	Bit Pair Mask	Displacement
OP Code; G, H, & I Fields								J Field		K Field

GIM2250

**SUMMARY:** If condition Control Store Address + K  $\rightarrow$  CR  
Else, execute next sequential instruction

**OPERATION:** This instruction is a conditional delayed jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by adding the Displacement (K FIELD) to the contents of the Control Register (DJIBO instruction address).

**NUMBER OF CYCLES:** DJIBO is a single-cycle instruction. The two instructions following DJIBO are always executed regardless of condition.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in L FIELD) if the first instruction following the delayed jump is a single word instruction.



**DELATED JUMP ON  
INDICATOR BIT  
BIT PAIR ONES MINUS**

**DJIBOM**

**MNEMONIC:** DJIBOM

**OP CODE:** B4

**FORMAT:**

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	0	1	0	0	Bit Pair Selctr	Bit Pair Mask	Displacement
OP Code; G, H, & I Fields								J Field		K Field

GIM2307

**SUMMARY:** If condition, Control Store Address - K → CR;  
else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional delayed jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to “Condition Selector” in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by subtracting the Displacement (K FIELD) from the contents of the Control Register (DJIBOM instruction address).

**NUMBER OF CYCLES:** DJIBOM is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJIBZ****DELAYED JUMP ON  
INDICATOR BIT PAIR ZEROS****DJIBZ****MNEMONIC:** DJIBZ**OP CODE:** 63**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	0	0	1	1	Bit Pair Selctr		Bit Pair Mask		Displacement			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2251

**SUMMARY:** If condition, Control Store Address + K  $\Rightarrow$  CR;  
Else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional delayed jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by adding the Displacement (K FIELD) to the contents of the Control Register (DJIBZ instruction address).

**NUMBER OF CYCLES:** DJIBZ is a single-cycle instruction. The two instructions following DJIBZ are always executed regardless of condition.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

DJIBZM

DELAYED JUMP ON  
INDICATOR BIT  
PAIR ZEROS MINUS

DJIBZM

MNEMONIC: DJIBZM

OP CODE: B5

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	0	1	0	1	Bit Pair Selctr		Bit Pair Mask		Displacement			
OP Code; G, H, & I Fields								J Field			K Field				

GIM2208

**SUMMARY:** If condition, Control Store Address - K → CR; else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional delayed jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to “Condition Selector” in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by adding the Displacement (K FIELD) to the contents of the Control Register (DJIBZM instruction address).

**NUMBER OF CYCLES:** DJIBZM is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJOR**      **DELAYED JUMP ON REGISTER****DJOR**

MNEMONIC: DJOR

OP CODE: 2D

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	0	1	1	0	1	Control			Source RSU				
	OP Code; G, H, & I Fields								J Field			K Field				

GIM2253

SUMMARY: (RK — RH) → CR

**OPERATION:** This instruction is a delayed unconditional jump. A halfword of data is transferred from the RSU addressed by the K FIELD to the Control Register, the most significant sixteen bits are not used. The contents of the RSU remain unchanged. The previous contents of the Control Register are lost. The two instructions following this instruction are executed before the jump is taken. DJOR is also used to set or reset breakpoints.

The J FIELD is used to specify control information relating to Breakpoints:

J05: This bit is used to enable setting or resetting breakpoints. With this bit a zero, J06 has no effect.

J06: If J05 is a one and J06 is a zero, then any breakpoint at the ISU location specified by the right halfword in RSU-K will be reset. If J05 is a one and J06 is a one, then a Breakpoint will be set at the ISU location specified by the right halfword in RSU-K.

Note: If this instruction is used to set or reset a Breakpoint, then the jump caused by the DJOR can be voided by programming an unconditional immediate jump after the DJOR.

**NUMBER OF CYCLES:** DJOR is a single-cycle instruction. The two instructions following DJOR are executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJPMBN****DELAYED JUMP ON  
PMBUS NEGATIVE****DJPMBN****MNEMONIC:** DJPMBN**OP CODE:** 3E**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J & K Fields							

GIM2297

**SUMMARY:** (PM BUS - RH)/ → CR

**OPERATION:** The least significant 16 bits of information that are on the PM Bus at X1 time during the execution of the DJPMB instruction will be inverted (one's complemented) and loaded into the CR.

**NUMBER OF CYCLES:** DJPMBN is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJRM    DELAYED JUMP RELATIVE MINUS    DJRM****MNEMONIC:** DJRM**OP CODE:** 6F**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	1	0	1	1	1	1	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2257

**SUMMARY:** Control Store Address - JK → CR

**OPERATION:** This instruction causes a delayed program jump in the positive direction. The jump address is formed by binarily subtracting the J K FIELD from the Control Register contents (DJRM instruction address). The maximum jump is 255 consecutive address locations. A Carryout of the sixteenth bit on the addition is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** DJRM is a single-cycle instruction. The two instructions following DJRM are executed before the jump is taken.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJRP**      DELAYED JUMP RELATIVE PLUS      **DJRP**

**MNEMONIC:** DJRP

**OP CODE:**      6E

**FORMAT:**      16 15 14 13 12 11 10 9    8 7 6 5 4 3 2 1

0	1	1	0	1	1	1	0	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2256

**SUMMARY:**      Control Store Address + JK → CR

**OPERATION:** This instruction causes a delayed program jump in the positive direction. The jump address is formed by binarily adding the J K FIELD to the Control Register contents (DJRP instruction address). The maximum jump is 255 consecutive address locations. A Carryout of the sixteenth bit on the addition is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** DJRP is a single-cycle instruction. The two instructions following DJRP are executed before the jump is taken.

**EFFECT ON INDICATOR ARRAY:**    None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DJTNZ****DELAYED JUMP ON TALLY  
NOT ZERO****DJTNZ****MNEMONIC:** DJTNZ**OP CODE:** 90**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	0	0	0	0	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2270

**SUMMARY:** If Tally not equal 0, Control Store Address - JK  
 $\rightarrow$  CR (T) - 1  $\rightarrow$  T  
 Else execute next sequential instruction.

**OPERATION:** This instruction is a conditional delayed relative jump with a negative displacement. The Tally Register is tested, if not equal to zero, the J and K FIELDS are binarily subtracted from the Control Store Address of the DJTNZ instruction and then transferred to the Control Register. The Tally Register is decremented by one. The previous contents of the Control Register are lost. The two instructions following this instruction are executed before the jump is taken.

If the Tally Register is zero the jump is not taken and the Tally Register is not decremented.

**NUMBER OF CYCLES:** DJTNZ is a single-cycle instruction. The two instructions following DJTNZ are executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed jump instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.



DRIBO

DELAYED RETURN ON  
INDICATOR BIT PAIR ONES

DRIBO

MNEMONIC: DRIBO

OP CODE: 65

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0 1 1 0 0 1 0 1								0	Jump Register		Bit Pair Selctr		Bit Pair Mask		
OP Code; G, H, & I Fields								J Field				K Field			

GIM2260

SUMMARY: If condition (JRJ) → CR  
Else execute next sequential instruction

OPERATION: This instruction is a conditional immediate jump. Bits 04,03 of the K FIELD select the Indicator Bit Pair to be compared against the Mask, bits 02,01 of the K FIELD. Refer to “Condition Selector” in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by transferring the contents of Jump Register specified by the J FIELD to the Control Register.

NUMBER OF CYCLES: DRIBO is a single-cycle instruction. The two instructions following DRIBO will be executed regardless of the condition.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: The second instruction following a delayed return instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**DRIBZ****DELAYED RETURN ON  
INDICATOR BIT PAIR ZEROS****DRIBZ****MNEMONIC:** DRIBZ**OP CODE:** 67

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	1	0	0	1	1	1	0	Jump Register			Bit Pair Selctr		Bit Pair Mask	
	OP Code; G, H, & I Fields								J Field			K Field				

GIM2261

**SUMMARY:** If condition (JRJ) → CR  
Else execute next sequential instruction

**OPERATION:** This instruction is a conditional immediate jump. Bits 04,03 of the K FIELD select the Indicator Bit Pair to be compared against the Mask, bits 02,01 of the K FIELD. Refer to "Condition Selector" in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by transferring the contents of Jump Register specified by the J FIELD to the Control Register.

**NUMBER OF CYCLES:** DRIBZ is a single-cycle instruction. The two instructions following DRIBZ will be executed regardless of the condition.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The second instruction following a delayed return instruction must not be a two word instruction (one requiring the trailing literal in the L FIELD) if the first instruction following the delayed jump is a single word instruction.

**EAF**      **EXTENDED ARITHMETIC FUNCTION**      **EAF****MNEMONIC:** EAF**OP CODE:**      12

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	1	0	0	1	0	Byte Literal							
	OP Code; G, H, & I Fields								J & K Fields							

GIM2293

**SUMMARY:**      JK → Extended Arithmetic Chip as an implicit transfer out to ERU63.

**OPERATION:** The instruction performs a transfer out to ERU63 (implicitly addressed by the EAF instruction). The value transferred is the J and K literal. The literal is placed in the least significant eight bits of the thirty-two bit word.

**NUMBER OF CYCLES:** EAF is a single-cycle instruction although contention on the PM Bus may halt the Processor until the PM Bus becomes available.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I7.

**PROGRAMMING CONVENTIONS:** None

**F****FETCH****F****MNEMONIC:** F**OP CODE:** 04**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	0	0	0	1	0	0	0	0	0	0	Source RSU			
OP Code; G, H, & I Fields								Not Used; J Field				K Field			

GIM2137

**SUMMARY:** (RK) → PM Bus

**OPERATION:** A virtual fetch (if AT is on) or a real fetch (if AT is off) is initiated from local memory using the address in the RSU specified by the K FIELD.

**NUMBER OF CYCLES:** F is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The F instruction must be followed by the RCV instruction to receive the data from the PM Bus.

**FL**

**FETCH LITERAL**

**FL**

MNEMONIC: FL

OP CODE: 08-0B

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	0	1	0	Literal Addr						0	0	0	0
	OP Code; G & H Fields						I & J Fields						K Field; Not Used			

GIM2141A

SUMMARY: 1 → PMBUS 24-10  
0 → PMBUS 09  
I,J → PMBUS 08-03  
O → PMBUS 02,01

OPERATION: A real memory fetch is initiated from local memory using the address formed by concatenating ones to the right-justified literal in the I and J FIELDS. The real address is a word address (0 mod 4) accessing the first 64-words of Local Memory Scratch Pad.

NUMBER OF CYCLES: The Fetch Literal operation is a single-cycle operation.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: The FL instruction must be followed by the RCV instruction to receive the data from the PM Bus.

**FR****FETCH REAL****FR****MNEMONIC:** FR**OP CODE:** 03

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	0	0	0	1	1	0	0	0	0	Source RSU			
	OP Code; G, H, & I Fields								Not Used; J Field				K Field			

GIM2136

**SUMMARY:** (RK) → PM Bus

**OPERATION:** A Real Fetch is initiated from Local Memory using the address in the rightmost three bytes of the RSU specified by the K FIELD.

**NUMBER OF CYCLES:** The Fetch Real operation is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The FR instruction must be followed by the RCV instruction to receive the data from the PM Bus.

JFA

JUMP ON FIELD ARRAY

JFA

MNEMONIC: JFA

OP CODE: 68

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2267

SUMMARY: If overflow, MARS OVERFLOW  
ADDRESS → CR;  
Else, execute next sequential instruction.

OPERATION: This instruction tests the MARS Field Overflow Flags. If any of the flags are on and the contents of the Tally Register are not equal to zero, the jump takes place. The jump address is formed by concatenating the most significant eight bits of Jump Register 7 with the Field Array vector. The final address is:

- Bits 16-9 → JRJ 7 (bits 16-9)
- Bit 8 → MARS Direction Flag
- Bit 7 → MARS 7 Overflow
- Bit 6 → MARS 6 Overflow
- Bit 5 → MARS 5 Overflow
- Bit 4 → MARS 4 Overflow
- Bit 3-1 → 0

NUMBER OF CYCLES: JFA is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met the two instructions following JFA are not executed.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**JFAL****JUMP ON FIELD ARRAY LINK****JFAL****MNEMONIC:** JFAL**OP CODE:** 69**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2268A

**SUMMARY:** If overflow, MARS OVERFLOW ADDRESS → CR, and next instruction address → JR6 (Link);  
Else, execute next sequential instruction.

**OPERATION:** This instruction tests the MARS Field Overflow Flags. If any of the flags are on and the contents of the Tally Register are not equal to zero, then the jump takes place. The jump address is formed by concatenating the most significant eight bits of Jump Register 7 with Field Array vector. The final address is:

Bits 16-9 → JRJ 7 (bits 16-9)  
 Bit 8 → MARS Direction Flag  
 Bit 7 → MARS 7 Overflow  
 Bit 6 → MARS 6 Overflow  
 Bit 5 → MARS 5 Overflow  
 Bit 4 → MARS 4 Overflow  
 Bit 3-1 → 0

The contents of the IAR at the time JFAL is in the execution stage of the pipeline are transferred to Jump Register 6 as the program Link Address.

**Note:** If the JFAL instruction is located immediately after a delayed jump instruction which takes, the Link established in JR6 will be the instruction address dictated by the logical jump sequence.

**NUMBER OF CYCLES:** JFAL is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met, the two instructions following JFAL are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



**JIBO****JUMP ON INDICATOR BIT  
PAIR ONES****JIBO****MNEMONIC:** JIBO**OP CODE:** 6A**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	1	0	1	0	1	0	Bit Pair Selctr	Bit Pair Mask	Displacement
OP Code; G, H, & I Fields								J Field		K Field

GIM2262

**SUMMARY:** If condition, Control Store Address + K → CR;  
Else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by adding the Displacement (K FIELD) to the contents of the control Register (JIBO instruction address).

**NUMBER OF CYCLES:** JIBO is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met, the two instructions following JIBO are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**JIBOL****JUMP ON INDICATOR BIT  
PAIR ONES LONG****JIBOL**

MNEMONIC: JIBOL

OP CODE: B6

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	0	1	1	0	Bit Pair Selctr		Bit Pair Mask		0	0	0	0
OP Code; G, H, & I Fields								J Field			K Field; Not Used				
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Literal															
L Field															

GIM2309

**SUMMARY:** If condition L  $\rightarrow$  CR; else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate jump and can be used to jump to any location in Control Store. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by transferring the L FIELD to the Control Register.

**NUMBER OF CYCLES:** JIBOL is a three-cycle instruction if the condition is met and two cycles if not. If the condition is met the instruction following JIBOL is not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



**JIBZ****JUMP ON INDICATOR BIT  
PAIR ZEROS****JIBZ****MNEMONIC:** JIBZ**OP CODE:** 6B**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	1	0	1	1	Bit Pair Selctr		Bit Pair Mask		Displacement			
OP Code; G, H, & I Fields								J Field							

GIM2263

**SUMMARY:** If condition, Control Store Address + K  $\rightarrow$  CR;  
Else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by adding the Displacement (K FIELD) to the contents of the control Register (JIBZ instruction address).

**NUMBER OF CYCLES:** JIBZ is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met, the two instructions following JIBZ are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

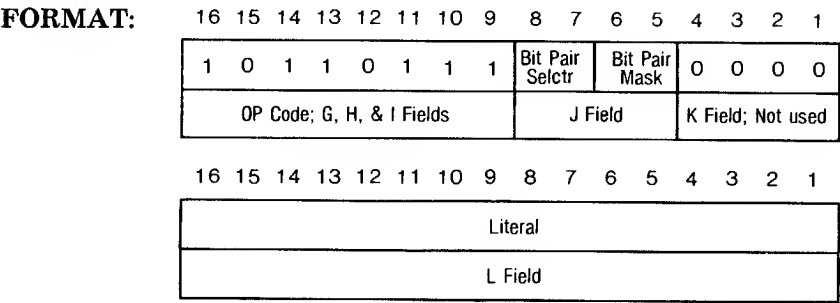
JIBZL

JUMP ON INDICATOR BIT  
PAIR ZEROS LONG

JIBZL

MNEMONIC: JIBZL

OP CODE: B7



GIM2310

SUMMARY: If condition, L → CR; else, execute next sequential instruction.

OPERATION: This instruction is a conditional immediate jump and can be used to jump to any location in Control Store. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to “Condition Selector” in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by transferring the L FIELD to the Control Register.

NUMBER OF CYCLES: JIBZL is a three-cycle instruction if the condition is met and a two-cycle instruction if the condition is not met. If the condition is met the instruction following JIBZL is not executed.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**JIBZM****JUMP ON INDICATOR BIT  
PAIR ZEROS MINUS****JIBZM****MNEMONIC:** JIBZM**OP CODE:** B3**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	0	0	1	1	Bit Pair Selctr	Bit Pair Mask	Displacement
OP Code; G, H, & I Fields								J Field		K Field

GIM2306

**SUMMARY:** If condition, Control Store Address - K → CR;  
else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate jump. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by subtracting the Displacement (K FIELD) from the contents of the Control Register (JIBZM instruction address).

**NUMBER OF CYCLES:** JIBZM is a three-cycle instruction if the condition is met and a one-cycle if the condition is not met. If the condition is met the two instructions following JIBZM are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**JIOL****JUMP ON INDICATOR ONES  
LINKAGE****JIOL****MNEMONIC:** JIOL**OP CODE:** 70-7F**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0 1 1 1				Literal				0	Jump Register		Literal				
OP Code; G Field				H & I Fields				J Field				K Field			

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Literal															
L Field															

GIM2265

**SUMMARY:** If condition,  $L \Rightarrow$  CR and next instruction address  $\Rightarrow$  JRJ; else, execute next sequential instruction:

**OPERATION:** This instruction compares the H, I, K FIELDS bit for bit against the Indicator Array. If any one bit in the H, I, K FIELDS matches a one bit in the Indicator Array the L FIELD is transferred to the Control Register and the next instruction address (normally JIOL address + 2) is transferred to the Jump Register addressed by the J FIELD (Link). The previous contents of the Jump Register are lost.

**NUMBER OF CYCLES:** JIOL is a three-cycle instruction if the condition is met and a two-cycle instruction if the condition is not met. If the condition is met, the instruction following JIOL is not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**JIZL****JUMP ON INDICATOR ZEROS  
LINKAGE****JIZL****MNEMONIC:** JIZL**OP CODE:** 80-8F**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	Literal				0	Jump Register			Literal			
OP Code; G Field				H & I Fields				J Field			K Field				

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Literal															
L Field															

GIM2266

**SUMMARY:** If condition,  $L \rightarrow CR$  and next instruction address  $\rightarrow JRJ$ ; else, execute next sequential instruction.

**OPERATION:** This instruction compares the H, I, K FIELDS bit for bit against the Indicator Array. If any one bit in the H, I, K FIELDS matches a Zero bit in the Indicator Array the L FIELD is transferred to the Control Register and the next instruction address (normally JIZL address + 2) is transferred to the Jump Register addressed by the J FIELD (Link). The previous contents of the jump register are lost.

**NUMBER OF CYCLES:** JIZL is a three-cycle instruction if the condition is met and a two-cycle instruction if the condition is not met. If the condition is met, the instruction following JIZL is not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



**JMPD****NVM DESCRIPTOR JUMP****JMPD****MNEMONIC:** JMPD**OP CODE:** 3B**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	1	0	1	1	0	0	0	0	Source RSU			
OP Code; G, H, & I Fields								J Field; Not Used				K Field			

GIM2269

**SUMMARY:** (RK – BO) → SUR1 1B – 09  
 Execution Address → CR

**OPERATION:** The Primary Setup Register (SUR1 16-09) is loaded from Byte 0 of the RSU specified by the K FIELD of the instruction. The setup hardware uses the Descriptor ID Field (SUR1 16-14) to form the Descriptor Execution Address.

The Execution Address is loaded into the CR and a delayed jump is initiated if (SUR1 12-09) is not equal to zero. The Execution Address is loaded into the CR and an immediate jump is initiated if (SUR1 12-09) is equal to zero.

**NUMBER OF CYCLES:** JMPD is a two-cycle instruction. SUR1 is loaded during the first cycle and the CR during the second. The immediate jump version of JMPD is, therefore, effectively a four-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

# JMPIA

## IBM SETUP JUMP A

# JMPIA

MNEMONIC: JMPA

OP CODE: 32

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	0	0	1	0	0	Jump Register	Source RSU
OP Code; G, H, & I Fields									J Field	K Field

GIM2241

SUMMARY: (RK - H) → SUR1, SP PTRS (IF BCT/+M780)  
(JRJ) → CR if BCT  
SETUP ADDR → CR if BCT/.M7B2  
M7OF ADDR → CR if BCT/.M7B0

**OPERATION:** If an instruction is present (MARS7 Byte Pointer = 1,0), and BCT is off, the Primary Setup Register (SUR1 16-01) is loaded from the right halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must designate RSU15). The Tally Copy Register, SUR5 08-01, is loaded from the least significant eight bits of the halfword. Formats that include a Tally value will load the Tally Register subsequently from SUR5. The setup hardware decodes the Command Op Code in SUR1 to determine the IBMVM Format type and loads the Scratch Pad Pointers.

If the BCT Indicator is true, the contents of the Jump Register specified by the J FIELD will be transferred to the CR and a delayed jump will be initiated.

If the BCT Indicator is false and the MARS7 Byte Pointers are pointing at byte 2 of RSU-K (M7B2), the Setup Address is loaded into the CR and a delayed jump is initiated. MARS7 Byte Pointers are incremented by 2.

If BCT is false and the MARS7 Byte Pointers are pointing at byte 0 of RSU-K (M7B0), the MARS7 Overflow Fetch Address is loaded into the CR and a delayed jump is initiated. MARS7 Byte Pointers are not incremented.

**NUMBER OF CYCLES:** JMPA is a two-cycle instruction. SUR1 and the Pointers are loaded during the first cycle. The CR is loaded during the second cycle. The jump cycle calculations are based upon the CR loading in the second cycle.

***MICROINSTRUCTION SET***

**EFFECT ON INDICATOR ARRAY:**   None

**PROGRAMMING CONVENTIONS:** None

**JMPIB****IBM SETUP JUMP B****JMPIB****MNEMONIC:** JMPIB**OP CODE:** 33

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	1	0	0	1	1	0	0	0	0	Source RSU			
	OP Code; G, H, & I Fields								J Field; Not Used				K Field			

GIM2242

**SUMMARY:** (RK - H) → SUR1, SP POINTERS  
 SETUP ADDRESS → CR if RR/  
 EXECUTION ADDRESS → CR if RR

**OPERATION:** The Primary Setup Register (SUR1 16-01) is loaded from the left halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must designate RSU15). The Tally Copy Register, SUR5 bits 08-01, is loaded from the least significant eight bits of the halfword. Formats that include a Tally value will subsequently load the Tally Register from SUR5. The setup hardware decodes the Command Op Code in SUR1 to determine the IBMVM Format type and loads the Scratch Pad Pointers.

The Mask field in the IBM instruction (SUR1 bits 08-05) is compared against the Condition Code Bits in the Virtual Indicator Array (VIA 02,01). If a match occurs then the Condition Code Match Bit (I6) in the Indicator Array will be set during the cycle following the JMPIB. If a match does not occur, or a match occurs but the IBM instruction format is RR and the R2 field is all zeros, then I6 will be reset.

If the Format is not the RR type, the Setup Address is loaded into the CR and an immediate (delayed for RX) jump is initiated.

If the IBM Format is the RR type, the Execution Address is loaded into the CR and a Delayed jump is initiated.

**NUMBER OF CYCLES:** JMPIB is a two-cycle instruction. SUR1 and the Pointers are loaded during the first cycle. The CR is loaded during the second cycle. The jump cycle calculations are based on the CR loading in the second cycle. Therefore, the immediate jump version of JMPIB is effectively a four-cycle instruction.

## *MICROINSTRUCTION SET*

**EFFECT ON VIRTUAL INDICATOR ARRAY:** This instruction affects Virtual Indicator I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately following JMPIB should not test Virtual Indicator I6.

# JMPIC

## IBM SETUP JUMP C

# JMPIC

MNEMONIC: JMPIC

OP CODE: 34

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2243

**SUMMARY:** EXECUTION ADDRESS → CR

**OPERATION:** The Mask field in the IBM instruction (SUR1 bits 08-05) is compared against the Condition Code Bits in the Virtual Indicator Array (VIA 02,01). If a match occurs, the Condition Code Match Bit (I6) in the Virtual Indicator Array will be set during the cycle following the JMPIC. If a match does not occur, or a match occurs but the IBM instruction format is **RR** and the R2 field is all zeros, Virtual Indicator I6 will be reset. The Execution Address is loaded into the CR and a delayed jump is initiated.

**NUMBER OF CYCLES:** JMPIC is a single-cycle instruction.

**EFFECT ON VIRTUAL INDICATOR ARRAY:** This instruction affects Virtual Indicator I6.

**PROGRAMMING CONVENTIONS:** None

**JMPNA**

## NVM SETUP JUMP A

**JMPNA**

MNEMONIC: JMPNA

OP CODE: 35

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	0	1	0	1	0		Register Jump	Source RSU
OP Code; G, H, & I Fields								J Field		K Field	

GIM2244

**SUMMARY:** (RK - H) → SUR1, SP PTRS (if BCT/+M7B0)  
 (JRJ) → CR if BCT  
 SETUP ADDR → CR if BCT/.M7B2.RR2/  
 EXEC ADDR → CR if BCT/.M7B2.RR2  
 M70F ADDR → CR if BCT/.M7B0

**OPERATION:** If an instruction is present (MARS7 Byte Pointers = 1,0) and BCT is off, the Primary Setup Register (SUR1 16-01) is loaded from the right halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must be designated RSU15). The Tally Copy Register, SUR5 bits 08-01, is loaded from the least significant eight bits of the halfword. Formats that include a Tally value will subsequently load the Tally Register from SUR5. The setup hardware decodes the Command Op Code in SUR1 to determine the NVM Format type and loads the Scratch Pad Pointers.

If the BCT Indicator is true, the contents of the jump Register specified by the J FIELD will be transferred to the CR and a delayed jump will be initiated.

If BCT is false and the MARS7 Byte Pointers are pointing at byte 2 of RSU-K (M7B2) and the NVM Format is not the RR2 type, the Setup Address is loaded into the CR and a delayed jump is initiated. MARS7 Byte Pointers are incremented by 2.

If BCT is false and the MARS7 Byte Pointers are pointing at byte 2 of RSU-K (M7B2) and the NVM Format is the RR2 type, the Execution Address is loaded into the CR and a delayed jump is initiated. MARS7 Byte Pointers are incremented by 2.

If BCT is false and the MARS7 Byte Pointers are pointing at byte 0 of RSU-K (M7B0), the MARS7 Overflow Fetch Address is loaded into the CR and a delayed jump is initiated. MARS7 Byte Pointers are not incremented.

**NUMBER OF CYCLES:** JMPNA is a two-cycle instruction. SUR1 and the Pointers are loaded during the first cycle. The CR is loaded during the second cycle. The jump cycle calculations are based upon the CR loading in the second cycle.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



JMPNB

NVM SETUP JUMP B

JMPNB

MNEMONIC: JMPNB

OP CODE: 36

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	0	1	1	0	0	0	0	0	Source RSU			
OP Code; G, H, & I Fields								J Field; Not Used				K Field			

GIM2245

SUMMARY: (RK - H) → SUR1, SP PTRS;  
SETUP ADDR → CR if RM or MM Formats  
EXEC ADDR → CR if RR1, RR2, or RI Formats

OPERATION: The Primary Setup Register (SUR1 16-01) is loaded from the left halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must designate RSU15). The Tally Copy Register, SUR5 bits 08-01, is loaded from the least significant eight bits of the halfword. Formats that include a Tally value will subsequently load the Tally Register from SUR5. The setup hardware decodes the Command Op Code in SUR1 to determine the NVM Format type and loads the Scratch Pad Pointers.

If the NVM Format is the MM type, the Setup Address is loaded into the CR and an immediate jump is initiated. If the format is the RM type, the Setup Address is loaded into the CR and a delayed jump is initiated.

If the NVM Format is the RR1 type, the Execution Address is loaded into the CR and a delayed jump is initiated.

If the NVM Format is the RR2 or RI types, the Execution Address is loaded into the CR and an immediate jump is initiated.

NUMBER OF CYCLES: JMPNB is a two-cycle instruction. SUR1 and the Pointers are loaded during the first cycle. The CR is loaded during the second cycle. The jump cycle calculations are based upon the CR loading in the second cycle. Therefore, the immediate jump version of JMPNB is effectively a four-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**JMPNC****NVM SETUP JUMP C****JMPNC****MNEMONIC:** JMPNC**OP CODE:** 37**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	0	1	1	1	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2246

**SUMMARY:** EXECUTION ADDRESS → CR if RM/+IN/  
INDIRECT ADDRESS → CR if RM AND IN

**OPERATION:** If the NVM Format is the RM type and the Indirection (In) Indicator is true in the instruction, the Indirection Address is loaded into the CR and a delayed jump is initiated. The Indirection Indicator, bit 04 in SUR1, is cleared following the execution of this instruction.

If the NVM Format is not the RM type or if the Indirection Indicator is false, the Execution Address is loaded into the CR and a delayed jump is initiated.

**NUMBER OF CYCLES:** JMPNC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# JMPVA

## VRX SETUP JUMP A

# JMPVA

MNEMONIC: JMPVA

OP CODE: 38

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	1	0	0	0	0	0	0	0	1	1	1	1
OP Code; G, H, & I Fields								J Field; Not Used				Source RSU K Field = 15			

GIM2247

**SUMMARY:** (RK - H) → SUR1  
SETUP A ADDRESS → CR

**OPERATION:** The Primary Setup Register (SUR1 16-01) is loaded from the left halfword of the RSU specified by the K FIELD (the K FIELD must designate RSU15).

The Setup “A” Address is loaded into the CR and a delayed jump is initiated.

**NUMBER OF CYCLES:** JMPVA is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# JMPVB

## VRX SETUP JUMP B

# JMPVB

**MNEMONIC: JMPVB**

OP CODE: 39

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0 0 1 1 1 0 0 1	0 0 0 0	1 1 1 1
OP Code; G, H, & I Fields	J Field; Not Used	Source RSU K Field = 15

GIM2248

**SUMMARY:**

(RK - H)	→ SUR1 (Rb)
	→ SUR5 (T)
SETUP "B"	ADDRESS → CR

**OPERATION:** The Primary Setup Register (SUR1 08-01) is loaded from the least significant eight bits of the left halfword of the RSU specified by the K FIELD (the K FIELD must be designated RSU15). The Tally Copy Register (SUR5 08-01) is loaded from the most significant eight bits.

The Setup "B" Address is loaded into the CR and a delayed jump is initiated.

**NUMBER OF CYCLES:** JMPVB is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

**PROGRAMMING CONVENTIONS:** None

**JMPVC**

VRX SETUP JUMP C

**JMPVC**

MNEMONIC: JMPVC

OP CODE: 3A

FORMAT:

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2249

**SUMMARY:** EXECUTION ADDRESS → CR  
 1 → TALLY 09 if T=0

**OPERATION:** The VRX Execution Address is loaded into the CR and a delayed jump is initiated.

If the contents of the Tally Register equals zero, the Virtual Tally should equal 256. A “one” is loaded into bit 09 of the Tally Register.

**NUMBER OF CYCLES:** JMPVC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**JOR****JUMP ON REGISTER****JOR****MNEMONIC:** JOR**OP CODE:** 2C**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	0	1	1	0	0	Control				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2252A

**SUMMARY:** (RK — RH) → CR

**OPERATION:** This instruction is an immediate unconditional jump. A halfword of data is transferred from the RSU addressed by the K FIELD to the Control Register, the most significant sixteen bits are not used. The contents of the RSU remain unchanged. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JOR is a three-cycle instruction. The two instructions following JOR are not executed.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**JPMBN      JUMP ON PM BUS NEGATIVE      JPMBN**

**MNEMONIC:**    JMPBN

**OP CODE:**      3C

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J & K; Not Used							

GIM2296

**SUMMARY:**    (PM Bus – RH)/ → CR

**OPERATION:**    The least significant 16 bits of information that are on the PM Bus at X1 time during the execution of the JPMB instruction will be inverted (one’s complemented) and loaded into the CR.

JMPBN is an immediate jump and therefore, the two instructions following JMPBN are not executed.

**NUMBER OF CYCLES:**    JMPBN is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:**    None

**PROGRAMMING CONVENTIONS:**    None

**JRM****JUMP RELATIVE MINUS****JRM****MNEMONIC:** JRM**OP CODE:** 6D**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	1	0	1	1	0	1	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2255

**SUMMARY:** Control Store Address - JK → CR

**OPERATION:** This instruction causes an immediate program jump in the positive direction. The jump address is formed by binarily subtracting the J and K FIELDS from the Control Register contents (JRM instruction address). The maximum jump is 255 consecutive address locations. A Carryout of the sixteenth bit on the addition is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JRM is a three-cycle instruction. The two instructions following JRM are not executed.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None



**JRMX    JUMP RELATIVE MINUS EXTERNAL    JRMX**

**MNEMONIC:**    JRMX

**OP CODE:**        B1

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	0	0	0	1	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2304

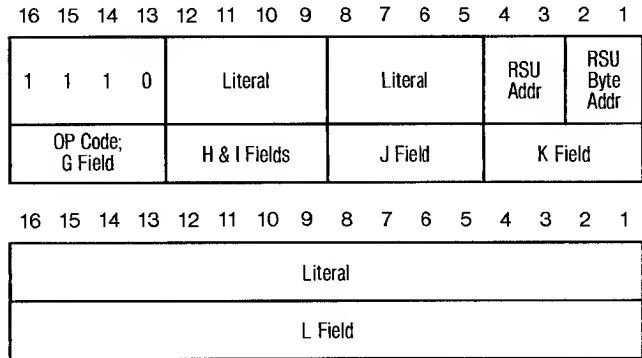
**SUMMARY:**        Control Store Address – JK → CR if external condition

**OPERATION:** This instruction causes an immediate program jump in the negative direction if the external condition is met. The condition tested is system dependent and in the absence of any system use for the external signal monitored by the Processor, the condition will never be met. The jump address is formed by binarily subtracting the J and K FIELDS from the Control Register contents (JRMX instruction address). The maximum jump is 255 consecutive address locations. A carryout of the sixteenth bit on the subtraction is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JRMX is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met the two instructions following JRMX are not executed.

**EFFECT ON INDICATOR ARRAY:**    None

**PROGRAMMING CONVENTIONS:**    None

**JRO****JUMP ON REGISTER ONES****JRO****MNEMONIC:** JRO**OP CODE:** EO-EF**FORMAT:**

GIM2264

**SUMMARY:** If condition,  $L \rightarrow CR$ ;  
Else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate jump and can be used to jump to any location in Control Store. During execution the H, I, J FIELDS are compared bit for bit against the byte addressed by the K FIELD. If all one bits in the H, I, J FIELDS match corresponding one bits in the Byte addressed by the K FIELD, the L FIELD is transferred to the Control Register. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JRO is a three-cycle instruction if the condition is met and a two-cycle instruction if the condition is not met. If the condition is met, the instruction following JRO is not executed.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

JRP

JUMP RELATIVE PLUS

JRP

MNEMONIC: JRP

OP CODE: 6C

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	1	0	1	1	0	0	Jump Length							
	OP Code; G, H, & I Fields								J & K Fields							

GIM2254

SUMMARY: Control Store Address + JK → CR

**OPERATION:** This instruction causes an immediate program jump in the positive direction. The jump address is formed by **binarily** adding the J K FIELDS to the Control Register contents (JRP instruction address). The maximum jump is 255 consecutive address locations. A Carryout of the sixteenth bit on the addition is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JRP is a three-cycle instruction. The two instructions following JRP are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**JRPX      JUMP RELATIVE PLUS EXTERNAL      JRPX****MNEMONIC:** JRPX**OP CODE:** B0

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	1	1	0	0	0	0	Jump Length							
	OP Code; G, H, & I Fields								J & K Fields							

GIM2303

**SUMMARY:** Control Store Address + JK → CR if external conditions

**OPERATION:** This instruction causes an immediate program jump in the positive direction if the external condition is met. The condition tested is system dependent and in the absence of any system use for the external signal monitored by the Processor, the condition will never be met.

The jump address is formed by binarily adding the J and K FIELDS to the Control Register contents (JPPX instruction address). The maximum jump is 255 consecutive address locations. A carryout of the sixteenth bit on the addition is lost. The previous contents of the Control Register are lost.

**NUMBER OF CYCLES:** JRPX is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met the two instructions following JRPX are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# JRZ

## JUMP ON REGISTER ZEROS

# JRZ

MNEMONIC: JRZ

OP CODE: FO-FF

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1 1 1 1	Literal	Literal	RSU Addr	RSU Byte Addr
OP Code: G Field	H, & I Fields	J Field	K Field	

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Literal															
L Field															

GIM2264AA

SUMMARY: If condition, L → CR;  
Else, execute next sequential instruction.

OPERATION: This instruction is a conditional immediate jump and can be used to jump to any location in Control Store. During execution the H, I, J FIELDS are compared bit for bit against the byte addressed by the K FIELD. If all one bits in the H, I, J FIELDS match corresponding zero bits in the Byte addressed by the K FIELD, the L FIELD is transferred to the Control Register. The previous contents of the Control Register are lost.

NUMBER OF CYCLES: JRZ is a three-cycle instruction if the condition is met and a two-cycle instruction if the condition is not met. If the condition is met, the instruction following JRZ is not executed.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**JTNZ****JUMP ON TALLY NOT ZERO****JTNZ****MNEMONIC:** JTNZ**OP CODE:** BA**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	0	1	0	Jump Length							
OP Code; G, H, & I Fields								J & K Fields							

GIM2313

**SUMMARY:** If T not equal 0, Control Store Address - JK → CR (T) - 1 → T; else, execute next sequential instruction.

**OPERATION:** This instruction is a conditional immediate relative jump with a negative displacement. The Tally Register is tested, if not zero, the J and K FIELDS are binarily subtracted from the Control Store Address of the JTNZ instruction and then transferred to the Control Register. The Tally Register is decremented by one. The previous contents of the Control Register are lost. The two instructions following this instruction are not executed.

If the Tally Register is zero the jump is not taken and the Tally is not decremented.

**NUMBER OF CYCLES:** JTNZ is a single-cycle instruction if the test condition is not met, and a three-cycle instruction if the test condition is met.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**LB**

**LOAD BYTE**

**LB**

**MNEMONIC:** LB

**OP CODE:** 59

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	1	0	0	1	Byte Literal							
OP Code; G, H, & I Fields								J & K Fields							

GIM2186

**SUMMARY:** J, K → R0, B0

**OPERATION:** The byte literal contained in the J, K FIELDS is loaded into byte 0 of RSU 0. The other three bytes of RSU 0 are not disturbed.

**NUMBER OF CYCLES:** LB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**LFA****LOAD, FETCH, AND AUGMENT****LFA****MNEMONIC:** LFA**OP CODE:** 05**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	0	0	0	1	0	1	Addr Dest RSU				Load Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2138

**SUMMARY:** (RK)  $\rightarrow$  PM Bus  
 (RK) + 4  $\rightarrow$  RJ

**OPERATION:** A virtual fetch (if AT is on) or a real fetch (if AT is off) is initiated from local memory using the address in the RSU specified by the K FIELD. The address in the K-RSU is augmented by four and loaded into the Address Register specified by the J FIELD.

**NUMBER OF CYCLES:** The Fetch and Augment operation is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The LFA instruction must be followed by the RCV instruction to receive data from the PM Bus.



**LFAL****LOAD, FETCH, AND AUGMENT  
LINKAGE****LFAL****MNEMONIC:** LFAL**OP CODE:** 06**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	0	0	0	1	1	0	Addr Dest RSU				Load Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2140

**SUMMARY:** (RK)  $\rightarrow$  PM Bus  
 (RK) + 4  $\rightarrow$  RJ

**OPERATION:** A virtual fetch (if AT is on) or a real fetch (if AT is off) is initiated from local memory using the address in the RSU specified by the K FIELD. The address in the K-RSU is augmented by four and loaded into the Address Register specified by the J FIELD. Linkage Protection is checked following address translation (if AT is on) in the DAT instead of read protection as on other fetches.

**NUMBER OF CYCLES:** The Load, Fetch, and Augment with Linkage operation is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The LFAL instruction must be followed by the RCV instruction to receive data from the PM Bus.

**LFD****LOAD, FETCH, AND DECREMENT****LFD****MNEMONIC:** LFD**OP CODE:** 07**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	0	0	0	1	1	1	Addr Dest RSU				Load Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2139

**SUMMARY:** (RK) → PM Bus  
 (RK) -4 → RJ

**OPERATION:** A virtual fetch (if AT is on) or a real fetch (if AT is off) is initiated from local memory using the address in the RSU specified by the K FIELD. The address in the K-RSU is decremented by four and loaded into the Address Register specified by the J FIELD.

**NUMBER OF CYCLES:** The Load, Fetch, and Decrement operation is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The LFD instruction must be followed by the RCV instruction to receive data from the PM Bus.

LINK

LOAD LINK ADDRESS

LINK

MNEMONIC: LINK

OP CODE: 92

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	0	0	1	0	0	Jump Register	Literal
OP Code; G, H, & I Fields										

GIM2290

SUMMARY: Control Store Address + K → JRJ

OPERATION: This instruction adds the Control Store Address (of LINK) to the literal in the K FIELD and stores the result in the Jump Register specified by the J FIELD.

NUMBER OF CYCLES: LINK is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: The J FIELD must not be set to a non-existent Jump Register address (must not be greater than 7).

**LINKM**      **LOAD LINK ADDRESS MINUS**      **LINKM**

MNEMONIC: LINKM

OP CODE: 61

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	1	0	0	0	0	1	0	Jump Register	Literal
OP Code; G, H, & I Fields									J Field	K Field

GIM2301

**SUMMARY:** Control Store Address - K → JRJ

**OPERATION:** This instruction decrements the Control Store Address (of LINKM) by the literal K and stores the result in the Jump Register specified by the J FIELD.

**NUMBER OF CYCLES:** LINKM is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The J FIELD must not be set to a non-existent Jump Register address (must not be greater than 7).

LLD

LOAD LEFT DIGIT

LLD

MNEMONIC: LLD

OP CODE: 5A

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	1	0	1	0	Dest RSU	Dest RSU Byte	Digit Literal
OP Code; G, H, & I Fields								J Field		K Field

GIM2188

SUMMARY: K → RJ – BLD

OPERATION: The digit literal contained in the K FIELD is loaded into the left digit in the byte of the RSU specified by the J FIELD. The right digit in the byte of the destination RSU is not disturbed.

NUMBER OF CYCLES: LLD is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**L RD****LOAD RIGHT DIGIT****L RD****MNEMONIC:** LRD**OP CODE:** 5B**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	1	0	1	1	Dest RSU	Dest RSU Byte	Digit Literal
OP Code; G, H & I Fields								J Field		K Field

GIM2187

**SUMMARY:** K → RJ - BRD

**OPERATION:** The digit literal contained in the K FIELD is loaded into the right digit in the byte of the RSU specified by the J FIELD. The left digit in the byte of the destination RSU is not disturbed.

**NUMBER OF CYCLES:** LRD is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

LRH

LOAD RIGHT HALFWORD

LRH

MNEMONIC: LRH

OP CODE: 5C

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	1	1	0	0	0	0	0	0	Dest RSU			
OP Code; G, H, & I Fields								Not Used; J Field				K Field			

16151413121110987654321

Literal															
L Field															

GIM2184

SUMMARY: L → RK - RH

OPERATION: The trailing literal (L FIELD) of the LRH instruction is loaded into the right halfword of the RSU specified by the K FIELD. The left halfword of the destination RSU is not disturbed.

NUMBER OF CYCLES: LRH is a two-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**LRHC****LOAD RIGHT HALFWORD  
CLEAR LEFT HALFWORD****LRHC****MNEMONIC: LRHC****OP CODE: 5D****FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	1	1	1	0	1	0	0	0	0	Dest RSU			
OP Code; G, H, & I Fields								Not Used; J Field				K Field			
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Literal															
L Field															

GIM2185

**SUMMARY:  $L \rightarrow RK - RH$** 

**OPERATION:** The trailing literal (L FIELD) of the LRHC instruction is loaded into the right halfword of the RSU specified by the K FIELD. The left halfword of the destination RSU is set to zero (cleared).

**NUMBER OF CYCLES:** LRHC is a two-cycle instruction.**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None



**LTRC**    **LOAD TALLY RIGHT CLEAR LEFT**    **LTRC**

MNEMONIC: LTRC

OP CODE:     58

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	1	1	0	0	0	Byte Literal							
	OP Code; G, H, & I Fields								J & K Fields							

GIM2189

**SUMMARY:**    J, K → TALLY Register - RB; 0 → TALLY Register-LB

**OPERATION:** The byte literal contained in the J, K FIELDS is loaded into the right half (byte) of the Tally Register. The left half of the Tally Register is set to zero (cleared).

**NUMBER OF CYCLES:** LTRC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction may affect Indicator I7.

**PROGRAMMING CONVENTIONS:** The LTRC instruction must not be executed immediately preceding any instruction that tests the contents of the Tally Register.

**LTS****LOAD TALLY FROM SETUP****LTS****MNEMONIC:** LTS**OP CODE:** 9B

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0
	OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2280

**SUMMARY:** 0's → TALLY 16 - 09 (SUR5 08 - 01) → TALLY 08 - 01

**OPERATION:** The Tally Registers eight least significant bits are loaded from the eight bits of SUR5. The eight most significant bits of the Tally Register are loaded with zeros.

**NUMBER OF CYCLES:** LTS is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** The LTS instruction must not be executed immediately preceding any instruction that tests the contents of the Tally Register.

**MII****MAP IBM INDICATORS****MII****MNEMONIC:** MII**OP CODE:** 54**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	1	0	0	Map Enables							
OP Code; G, H, & I Fields								J & K Fields							

GIM2281

**SUMMARY:** Machine Indicators mapped to IBM 370 Virtual Indicator Array.

**OPERATION:** The basic VLSI machine indicators are mapped to the IBM 370 Condition Code (Virtual Indicators) when specific Map Enables (J and K FIELDS) are set in the instruction.

I1 (L) is mapped to Virtual Indicator CC0 if J,K 01 is on.

I3 (G) is mapped to Virtual Indicator CC1 if J,K 02 is on.

I2 (E/) is mapped to Virtual Indicator CC0 if J,K 03 is on.

I4 (C) is mapped to Virtual Indicator CC1 if J,K 04 is on.

I4 (C) is mapped to Virtual Indicator CC1,0 if J,K 05 is on.

I5 (O) is mapped to Virtual Indicator CC1,0 if J,K 06 is on.

J05 and J06 are mutually exclusive. Also, when either J05 or J06 is on, J01-04 must be off.

**NUMBER OF CYCLES:** MII is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**MIN****MAP NVM INDICATORS****MIN****MNEMONIC:** MIN**OP CODE:** 55**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	1	0	1	Map Enables							
OP Code; G, H, & I Fields								J & K Fields							

GIM22B2

**SUMMARY:** Machine Indicators mapped to NVM Virtual Indicator Array.**OPERATION:** The basic VLSI machine indicators are mapped to the NVM Virtual Indicators when specific Map Enables (J and K FIELDS) are set in the instruction.

- I1 (L) is mapped to Virtual Indicator L if J,K 01 is on.
- I2 (E) is mapped to Virtual Indicator E if J,K 02 is on.
- I3 (G) is mapped to Virtual Indicator G if J,K 03 is on.
- I4 (C) is mapped to Virtual Indicator O if J,K 04 is on.
- I4/(C/) is mapped to Virtual Indicator O if J,K 05 is on.

**NUMBER OF CYCLES:** MIN is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

MIV

MAP VRX INDICATORS

MIV

MNEMONIC: MIV

OP CODE: 56

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	1	0	1	1	0	Map Enables							
	OP Code; G, H, & I Fields								J & K Fields							

GIM2283

SUMMARY: Machine Indicators mapped to VRX Virtual Indicator Array.

OPERATION: The basic VLSI machine indicators are mapped to the VRX Virtual Indicators when specific Map Enables (J and K FIELDS) are set in the instruction.

- I1 (L) is mapped to Virtual Indicator L if J,K 01 is on.
- I2 (E) is mapped to Virtual Indicator E if J,K 02 is on.
- I3 (G) is mapped to Virtual Indicator G if J,K 03 is on.
- I4 (C) is mapped to Virtual Indicator O if J,K 04 is on.
- I5 (O) is mapped to Virtual Indicator O if J,K 05 is on.
- A zero is mapped to Virtual Indicator RI if J,K 06 is on.
- A one is mapped to Virtual Indicator RI if J,K 07 is on.

Note: 1J,K 06 and 07 are mutually exclusive, both cannot be on at the same time. J,K 04 and 05 are also mutually exclusive.

NUMBER OF CYCLES: MIV is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**MRR****MEMORY REFERENCE RETRY****MRR****MNEMONIC:** MRR**OP CODE:** 02**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	0	0	0	0	1	0	0	0	0	0	Source RSU			
OP Code; G, H, & I Fields								J Field; Not Used				K Field			

GIM2147

**SUMMARY:** (RK)  $\Rightarrow$  PM Bus  
 (RX)  $\Rightarrow$  PM Bus if S.R 16,15=0,1  
 where X = S.R 14-11

**OPERATION:** This instruction is used to re-execute virtual memory operations that were aborted due to address translation errors (DAT Interrupt). The contents of the RSU specified by the K FIELD are transferred via the PM Bus to the DAT logic as a Virtual Address.

If State Register bits 16,15=0,1 then the operation is a Virtual Store and data will be transferred over the PM Bus from the RSU specified by the State Register bits 14-11. The Write Tags will be supplied from State Register bits 10-07.

If State Register Bits 16,15 $\neq$ 0,1 then the operation is a virtual fetch with bits 16,15 controlling the mode of protection check to be performed (Read, Linkage or Execute).

**NUMBER OF CYCLES:** MRR is a single-cycle instruction, but when executing a store operation the PM Bus will be utilized for up to 4 cycles.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**RC****RESET CONTROLS****RC****MNEMONIC:** RC**OP CODE:** 95**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	0	1	0	1	Byte Literal							
OP Code; G, H, & I Fields								J & K Fields							

GIM2292

**SUMMARY:** (CA - LB) AND JK → CA - LB

**OPERATION:** The left byte in the Control Array #1 (bits 16-09) is logically ANDed with the inverted (logically negated) byte literal in the J and K FIELDS. The result replaces the left byte in the Control Array #1.

**NUMBER OF CYCLES:** RC is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** This instruction potentially affects Indicator I8.**PROGRAMMING CONVENTIONS:** None

# RCV

## RECEIVE FETCHED DATA

# RCV

**MNEMONIC:** RCV

**OP CODE:** 3D

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	1	1	0	1	Dest RSU				Control			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2284

**SUMMARY:** (PM Bus) → RJ or R(SR14 - 11)

**OPERATION:** A word (32 bits) of data is transferred from the PM Bus to RSU when that data is available. Until the data is available, the Processor pipeline is halted. A trap or an interrupt will also unlock the pipeline. If bit 2 in the K FIELD is off, the destination RSU is specified by the J FIELD. If bit 2 in the K FIELD is on, the destination RSU is specified by bits 14-11 of the State Register.

RCV is used in conjunction with Fetch instructions and TIES from Scratch Pad. On Virtual Fetches (AT on) the destination RSU must be odd-numbered.

If bit 1 in the K FIELD is on, the address of the Destination RSU (the J FIELD) will be loaded into bits 14-11 of the State Register provided AT is on.

**NUMBER OF CYCLES:** RCV is a multi-cycle instruction. The RCV instruction remains frozen in the pipeline until the DIE signal is asserted.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** K FIELD control bits 1 and 2 must never be set on together.



**RIBO**

**RETURN ON INDICATOR BIT  
PAIR ONES**

**RIBO**

**MNEMONIC:** RIBO

**OP CODE:** 64

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	0	1	0	0	0		Jump Register	Bit Pair Selctr		Bit Pair Mask		
OP Code; G, H, & I Fields									J Field			K Field			

GIM2258

**SUMMARY:** If condition (JRJ) → CR  
Else execute next sequential instruction

**OPERATION:** This instruction is a conditional immediate jump. Bits 04,03 of the K FIELD select the Indicator Bit Pair to be compared against the Mask, bits 02,01 of the K FIELD. Refer to “Condition Selector” in this chapter for an explanation of the Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The jump is formed by transferring the contents of Jump Register specified by the J FIELD to the Control Register.

**NUMBER OF CYCLES:** RIBO is a three-cycle instruction if the condition is met and a one-cycle instruction if not. If the condition is met the two instructions following RIBO are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**RIBZ****RETURN ON INDICATOR BIT  
PAIR ZEROS****RIBZ****MNEMONIC:** RIBZ**OP CODE:** 66

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	0	1	1	0	0	Jump Register		Bit Pair Selctr		Bit Pair Mask		
OP Code; G, H, & I Fields								J Field			K Field				

GIM2259

**SUMMARY:** If condition (JRJ) → CR  
Else execute next sequential instruction

**OPERATION:** This instruction is a conditional immediate jump. Bits 04,03 of the K FIELD select the Indicator Bit Pair to be compared against the Mask, bits 02,01 of the K FIELD. Refer to “Condition Selector” in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The jump is formed by transferring the contents of the Jump Register specified by the J FIELD to the Control Register.

**NUMBER OF CYCLES:** RIBZ is a three-cycle instruction if the condition is met and a one-cycle instruction if the condition is not met. If the condition is met, the two instructions following RIBZ are not executed.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

RIZ

RESET INDICATORS TO ZERO

RIZ

MNEMONIC: RIZ

OP CODE: 9F

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
	OP Code; G, H, & I Fields								J & K Fields; Not Used							

GIM2295

SUMMARY: 0 → I4, I5, I6

OPERATION: Indicators I4, I5 and I6 in the Indicator Array are set to zero.

NUMBER OF CYCLES: RIZ is a single-cycle operation.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I4, I5 and I6.

PROGRAMMING CONVENTIONS: None

**RTI**      **RESTORE FROM TRAPS/INTERRUPTS****RTI**

MNEMONIC: RTI

OP CODE:      91

FORMAT:      16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	0	0	0	1	Not Used	Control	Not Used	Control
OP Code; G, H, & I Fields								J Field		K Field	

GIM2289

SUMMARY:      (Restore FIFO) → CR

**OPERATION:** This instruction pops the top entry off the halted Restore FIFO and loads it into the CR. It is used in a restore sequence to partially restore the FIFO. Three RTI instructions are required to complete the sequence and restart the FIFO.

The first RTI instruction has the K FIELD set to zero. The second RTI instruction has bit 01 of the K FIELD set to a one which turns off the Trap Indicator (if it was on) and turns on the Normal Interrupt Enable. If another trap or interrupt is pending, then both the Trap Indicator and the Normal Interrupt enable will be appropriately set/reset during the next instruction. The third RTI has bit 02 of the K FIELD set to a one, which restarts the clocking of the FIFO and loads the skip count from bits 15,16 of Control Array 1 unless another interrupt or trap is pending.

In the Restore sequence the RTI instructions which have K02 set to a zero are executed with the FIFO already stopped. However, if an RTI instruction is executed with K02 equal to zero and the FIFO is being clocked, the execution of that RTI will halt the FIFO.

The J FIELD control bits 06 and 05 are used when the RTI is part of a Breakpoint Trap Routine which restores control to the instruction address at which the Breakpoint was detected.

**J05:** This bit is used to enable the reinforcing of the Breakpoint which caused the trap or the clearing of the Breakpoint. With this bit a zero, J06 has no effect and the same Breakpoint will be re-executed.

**J06:** If J05 is a one and J06 is a zero, then the Breakpoint at the ISU location specified by the top entry on the Restore FIFO will be cleared (reset). If J05 is a one and J06 is a one, then the Breakpoint at the specified ISU location will

be reinforced (set). RTI is effectively a delayed jump instruction.

**NUMBER OF CYCLES:** RTI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**S****STORE****S****MNEMONIC:** S**OP CODE:** 15**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	1	0	1	Write Tags				Addr Data RSUs			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2143

**SUMMARY:** (RKE) → PM Bus  
 (RKO) → PM Bus

**OPERATION:** A word (32 bits) of data is transferred from the Data (odd numbered) RSU specified by the K FIELD to local memory using the virtual address (AT on) or the real address (AT off) in the address (even numbered) RSU specified by the K FIELD.

The Memory Write Tags to be enabled are specified by the J FIELD:

- Write Tag for Byte 0 - J08
- Write Tag for Byte 1 - J07
- Write Tag for Byte 2 - J06
- Write Tag for Byte 3 - J05

If the entire J FIELD is set to zero, the Write Tags are supplied by the contents of the MARS6 Write Tag Register.

**NUMBER OF CYCLES:** SR is a single-cycle instruction, but the PM Bus is used for up to four cycles (for a partial virtual store). Therefore, the instructions following S will be suspended if they attempt to access the PM Bus while the store is completing.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**SA****STORE AND AUGMENT****SA****MNEMONIC:** SA**OP CODE:** 16**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	1	1	0	Write Tags				Addr RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2144

**SUMMARY:** (RKE) → PM Bus  
 (RKE) + 4 → RKE  
 (RKO) → PM Bus

**OPERATION:** A word (32 bits) of data is transferred from the Data (odd numbered) RSU specified by the K FIELD to local memory using the virtual address (AT on) or the real address (AT off) in the address (even numbered) RSU specified by the K FIELD. The address is augmented by four and loaded back into the Address RSU.

The Memory Write Tags to be enabled are specified by the J FIELD:

- Write Tag for Byte 0 - J08
- Write Tag for Byte 1 - J07
- Write Tag for Byte 2 - J06
- Write Tag for Byte 3 - J05

If the entire J FIELD is set to zero, the Write Tags are supplied by the contents of the MARS6 Write Tag Register.

**NUMBER OF CYCLES:** SA is a single-cycle instruction, but the PM Bus is used for up to four cycles (for a partial virtual store). Therefore the instructions following SA will be suspended if they attempt to access the PM Bus while the store is completing.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**SB****SUBTRACT BYTE****SB****MNEMONIC:** SB**OP CODE:** C7**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	0	1	1	1	Source Dest RSU	Source Dest RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2228

**SUMMARY:**  $(RJ - B) - (RK - B) \rightarrow RJ - B$ 

**OPERATION:** A byte from the RSU specified by the K FIELD is binarily subtracted from a byte from the RSU specified by the J FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, and I4.

**PROGRAMMING CONVENTIONS:** None



**SBC****SUBTRACT BYTE WITH CARRY****SBC**

MNEMONIC: SBC

OP CODE: C9

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	1	0	0	1	Source Dest RSU		Source Dest RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field			K Field				

GIM2229

SUMMARY:  $(RJ - B) - (RK - B) + C \rightarrow RJ - B$ 

**OPERATION:** A byte from RSU specified by the K FIELD is binarily subtracted from a byte from the RSU specified by the J FIELD. The Carry Indicator (I4) is binarily added to the least significant bit in the operation. The results replace the byte in the RSU specified by the J FIELD. A borrow as a result of the subtraction will set I4 OFF, no borrow will set I4 ON.

**NUMBER OF CYCLES:** SBC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, and I4.

**PROGRAMMING CONVENTIONS:** None

**SBL****SUBTRACT BYTE LITERAL****SBL****MNEMONIC:** SBL**OP CODE:** DE**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	1	1	1	1	0	Source Dest RSU	Source Dest RSU Byte	Digit Literal
OP Code; G, H, & I Fields								J Field		K Field

GIM2227

**SUMMARY:**  $(RJ - B) - K \rightarrow RJ - B$ 

**OPERATION:** A byte from RSU specified by the J FIELD is decremented by the digit literal in the K FIELD. The results replace the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SBL is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, and I4.

**PROGRAMMING CONVENTIONS:** None

**SC****SET CONTROLS****SC****MNEMONIC:** SC**OP CODE:** 94**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	0	1	0	0	Byte Literal							
OP Code; G, H, & I Fields								J & K Fields							

GIM2291

**SUMMARY:** (CA - LB) OR JK → CA - LB

**OPERATION:** The left byte in the Control Array #1 (bits 16-09) is logically ORed with the byte literal in the J and K FIELDS. The result replaces the left byte in the Control Array #1.

**NUMBER OF CYCLES:** SC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction potentially affects Indicator I8.

**PROGRAMMING CONVENTIONS:** None

**SCO**

**SET CARRY TO ONE**

**SCO**

**MNEMONIC:** SCO

**OP CODE:** 9E

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J & K Fields; Not Used							

GIM2294

**SUMMARY:** 1 → I4

**OPERATION:** The Carry Indicator (I4) in the Indicator Array is set to a one.

**NUMBER OF CYCLES:** SCO is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** None

# SD

## STORE AND DECREMENT

# SD

**MNEMONIC:** SD

**OP CODE:** 17

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	1	0	1	1	1	Write Tags				Addr RSU			
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2145A

**SUMMARY:** (RKE) → PM Bus  
 (RKE) -4 → RKE  
 (RKO) → PM Bus

**OPERATION:** A word (32 bits) of data is transferred from the Data (odd numbered) RSU corresponding to the even numbered RSU specified by the K FIELD to local memory using the virtual address (AT on) or the real address (AT off) in the Address RSU. The address is decremented by four and loaded back into the Address RSU.

The Memory Write Tags to be enabled are specified by the J FIELD:

- Write Tag for Byte 0 - J08
- Write Tag for Byte 1 - J07
- Write Tag for Byte 2 - J06
- Write Tag for Byte 3 - J05

If the entire J FIELD is set to zero, the Write Tags are supplied by the contents of the MARS6 Write Tag Register.

**NUMBER OF CYCLES:** SD is a single-cycle instruction, but the PM Bus is used for up to four cycles (for a partial virtual store). Therefore the instructions following SD will be suspended if they attempt to access the PM Bus while the store is completing.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**SETIA****IBM SETUP ASSIST A****SETIA****MNEMONIC:** SETIA**OP CODE:** 2E**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	0	1	1	1	0	Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2276

**SUMMARY:** (RK - H) → SUR2, SPTR2 DISPLACEMENT  
→ RSU - J

**OPERATION:** The Secondary Setup Register (SUR2 16-01) is loaded from either the left or the right halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must specify RSU15). Scratch Pad Pointer #2 is loaded from the "B" field of the Virtual Command.

The Displacement field in the Virtual Command is transferred to the RSU specified by the J FIELD. The MARS7 Byte Pointers are incremented by two.

**NUMBER OF CYCLES:** SETIA is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None .

**PROGRAMMING CONVENTIONS:** None

**SETNA**

NVM SETUP ASSIST A

**SETNA**

MNEMONIC: SETNA

OP CODE: 2F

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	0	1	1	1	1	Dest RSU				1	1	1	1
	OP Code; G, H, & I Fields								J Field				Source RSU K Field = 15			

GIM2278A

SUMMARY: (RK - H) → SUR2, SPTR2

**OPERATION:** The Secondary Setup Register (SUR2 16-01) is loaded from either the left or the right halfword of the RSU specified by the K FIELD as determined by the MARS7 Byte Pointers (the K FIELD must designate RSU15). Scratch Pad Pointer #2 is loaded except for RI Formats.

If the RI Format, the Virtual Indicators are matched against those bits that are true in the "C" FIELD of the Virtual Instruction (SUR1 04-01). If a match exists on any ones in the "C" FIELD, the internal Condition Code Match Indicator (I6) is set otherwise, it is reset. The Branch Offset (SUR2 16-01) is transferred to the RSU specified by the J FIELD. If bit 16 of SUR2 is a zero, then zeros are loaded into the upper half of RSU-J. If bit 16 is a one, then ones are loaded into the upper half of RSU-J. The MARS7 Byte Pointers are incremented by 2.

If the RM or MM Formats, the Displacement (SUR2 12-01) is transferred to the RSU specified by the J FIELD. The MARS7 Byte Pointers are incremented by 2.

If the RR Format is used and SETNA is executed, the results are predictable but not specified.

**NUMBER OF CYCLES:** SETNA is a single-cycle instruction.

**EFFECT ON VIRTUAL INDICATOR ARRAY:** This instruction affects Virtual Indicator I6.

**PROGRAMMING CONVENTIONS:** None

**SETSX****SETUP SIGN EXTENSION****SETSX****MNEMONIC:** SETSX**OP CODE:** 9A**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	1	0	1	0	0	0	0	0	Dest RSU			
OP Code; G, H, & I Fields								J Field; Not Used				K Field			

GIM2279

**SUMMARY:** (SUR2 16 - 01)  $\rightarrow$  RK 16 - 01, if SUR2 16=1 then 1  $\rightarrow$  RK 32 - 17; if SUR2 16=0 then 0  $\rightarrow$  RK 32 - 17

**OPERATION:** The contents of Setup Register #2 (SUR2 16-01) are transferred into the least significant sixteen bits of the RSU specified by the K FIELD. SUR2 bit 16 is tested. If SUR2 bit 16 is a zero, then zeros are transferred into the most significant sixteen bits of RSU-K. If SUR2 bit 16 is a one, then ones are transferred into the most significant sixteen bits of RSU-K.

**NUMBER OF CYCLES:** SETSX is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



SF

SUBTRACT FIELD

SF

MNEMONIC: SF

OP CODE: 41

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	0	0	0	1	1	0	0	1	1	0	1	1
OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2232

SUMMARY:  $(RJ - B) - (RK - B) + C \rightarrow R13$ ; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary

OPERATION: If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers is subtracted binarily from a byte from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers. The result is placed in the byte in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. Following the subtraction, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

NUMBER OF CYCLES: SF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I4.

PROGRAMMING CONVENTIONS: The instruction immediately preceding SF must not alter the Tally Register or the MARS Address Registers used by SF.

**SIBO**    SKIP ON INDICATOR BIT PAIR ONES    **SIBO**

MNEMONIC: SIBO

OP CODE:    B8

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	1	1	1	0	0	0	Bit Pair Selctr	Bit Pair Mask	0	0	0	0		
	OP Code; G, H, & I Fields								J Field			K Field; Not Used				

GIM2311

**SUMMARY:**    If condition, skip next sequential instruction;  
                      Else, execute next sequential instruction.

**OPERATION:** This instruction conditionally causes the next instruction in the pipeline to be skipped. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The skip is executed by voiding the next instruction in the pipeline.

**NUMBER OF CYCLES:** SIBO is a single-cycle instruction if the condition is not met.

**EFFECT ON INDICATOR ARRAY:**    None

**PROGRAMMING CONVENTIONS:** SIBO must only be used to skip single word instructions.

**SIBZ**    SKIP ON INDICATOR BIT PAIR ZEROS    **SIBZ**

MNEMONIC: SIBZ

OP CODE:     B9

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	1	1	0	0	1	Bit Pair Selctr	Bit Pair Mask	0	0	0	0		
OP Code; G, H, & I Fields								J Field			K Field; Not Used				

GIM2312

**SUMMARY:**     If condition, skip next sequential instruction;  
                  Else, execute next sequential instruction.

**OPERATION:** This instruction conditionally causes the next instruction in the pipeline to be skipped. Bits 08,07 of the J FIELD select the Indicator Bit Pair to be compared against the Mask bits 06,05 of the J FIELD. Refer to “Condition Selector” in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

                This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The skip is executed by voiding the next instruction in the pipeline.

**NUMBER OF CYCLES:** SIBZ is a single-cycle instruction if the condition is not met.

**EFFECT ON INDICATOR ARRAY:**    None

**PROGRAMMING CONVENTIONS:** SIBZ must only be used to skip single word instructions.

**SL****STORE LITERAL****SL****MNEMONIC:** SL**OP CODE:** 18-1B

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	1	1	0	Literal Addr						Data RSU			
	OP Code; G & H Fields						I & J Fields						K Field			

GIM2146A

**SUMMARY:**

- 1 → PMBUS 24-10
- 0 → PMBUS 09
- I,J → PMBUS 08-03
- O → PMBUS 02, 01
- (RK) → PM Bus

**OPERATION:** A word (32 bits) of data is transferred from the RSU specified by the K FIELD to local memory using the real memory address formed by concatenating ones to the right-justified literal in the I, J FIELDS. The real address is a word address (0 mod 4) accessing the first 64-words of local memory.

**NUMBER OF CYCLES:** SL is a single-cycle instruction, but the PM Bus is used for two cycles. Therefore, the instruction following SL will be suspended if it attempts to access the PM Bus.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**SPDB**    SUBTRACT PACKED DECIMAL BYTE    **SPDB**

MNEMONIC:    SPDB

OP CODE:       CF

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	1	1	1	1	Source Dest RSU	Source Dest RSU Byte	Source RSU	Source RSU Byte				
OP Code; G, H, & I Fields								J Field		K Field					

GIM2230

**SUMMARY:**        (RJ - B) - (RK - B) → RJ - B

**OPERATION:**    A byte from the RSU specified by the K FIELD is subtracted packed decimally from a byte from the RSU specified by the J FIELD. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:**    SPDB is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:**    This instruction affects Indicator I1, I2, I3, I4 and I5.

**PROGRAMMING CONVENTIONS:**    None

# SPDBC SUBTRACT PACKED DECIMAL BYTE WITH CARRY SPDBC

**MNEMONIC:** SPDBC

**OP CODE:** D1

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	1	0	0	0	1	Source Dest RSU	Source Dest RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2231

**SUMMARY:**  $(RJ - B) - (RK - B) + C \rightarrow (RJ - B)$

**OPERATION:** A byte from the RSU specified by the K FIELD is subtracted packed decimally from a byte from the RSU specified by the J FIELD; the result is added to the Carry Bit (I4) from a previous instruction. The result replaces the byte in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SPDBC is a single-cycle operation.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, I4 and I5.

**PROGRAMMING CONVENTIONS:** None

**SPDF SUBTRACT PACKED DECIMAL FIELD SPDF**

MNEMONIC: SPDF

OP CODE: 47

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	0	0	1	1	1	1	0	0	1	1	0	1	1
	OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2233

**SUMMARY:**  $(R_J - B) - (R_K - B) + C \rightarrow R_{13}$ ; decrement Byte Pointer; decrement Tally Register; set M#OF if word boundary

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers is subtracted packed decimally from a byte from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers. The result is placed in the byte in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. Following the subtraction, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** SPDF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects I4, and I5.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding SPDF must not alter the Tally Register or the MARS Address Registers used by SPDF.

**SR****STORE REAL****SR****MNEMONIC:** SR**OP CODE:** 14

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	1	0	1	0	0	Write Tags				Addr Data RSUs			
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2142

**SUMMARY:** (RKE) → PM Bus  
 (RKO) → PM Bus

**OPERATION:** A word (32 bits) of data is transferred from the Data (odd numbered) RSU specified by the K FIELD to local memory using the real address in the address (even numbered) RSU specified by the K FIELD.

The Memory Write Tags to be enabled are specified by the J FIELD:

- Write Tag for Byte 0 - J08
- Write Tag for Byte 1 - J07
- Write Tag for Byte 2 - J06
- Write Tag for Byte 3 - J05

If the entire J FIELD is set to zero, the Write Tags are supplied by the contents of the MARS6 Write Tag Register.

**NUMBER OF CYCLES:** SR is a single-cycle instruction, but the PM Bus is used for up to three cycles (for a partial real store). Therefore, the instructions following SR will be suspended if they attempt to access the PM Bus while the store is completing.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



SRBO

SKIP ON REGISTER BIT PAIR ONES

SRBO

MNEMONIC: SRBO

OP CODE: C4

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	0	1	0	0	Bit Pair Selctr	Bit Pair Mask	Source RSU	Source RSU Byte				
OP Code; G, H, & I Fields								J Field		K Field					

GIM2273

SUMMARY: If condition, skip next sequential instruction;  
Else, execute next sequential instruction.

OPERATION: This instruction conditionally causes the next instruction in the pipeline to be skipped. The K FIELD selects the RSU byte to be tested. Bits 08,07 of the J FIELD select the Register Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to “Condition Selector” in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The skip is executed by voiding the next instruction in the pipeline.

NUMBER OF CYCLES: SRBO is a single-cycle instruction if the condition is not met.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: SRBO must only be used to skip single-cycle instructions.

**SRBZ** SKIP ON REGISTER BIT PAIR ZEROS **SRBZ**

MNEMONIC: SRBZ

OP CODE: C5

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	0	1	0	1	Bit Pair Selctr	Bit Pair Mask	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2274

**SUMMARY:** If condition, skip next sequential instruction;  
Else, execute next sequential instruction.

**OPERATION:** This instruction conditionally causes the next instruction in the pipeline to be skipped. The K FIELD selects the RSU byte to be tested. Bits 08,07 of the J FIELD select the Register Bit Pair to be compared against the Mask, bits 06,05 of the J FIELD. Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for zeros in the Test Bits. A logical zero satisfies the test. The skip is executed by voiding the next instruction in the pipeline

**NUMBER OF CYCLES:** SRBZ is a single-cycle instruction if the condition is not met.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** SRBZ must only be used to skip single-cycle instructions.

**SRB30** SKIP ON REGISTER BYTE 3 ONES **SRB30**

MNEMONIC: SRB30

OP CODE: 60

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	1	0	0	0	0	0	Bit Pair Selection	Bit Pair Mask	Source RSU					
OP Code; G, H, & I Fields								J Field			K Field				

GIM1001

**SUMMARY:** If condition, skip next sequential instruction;  
Else, execute next sequential instruction.

**OPERATION:** This instruction conditionally causes the next instruction in the pipeline to be skipped. The K FIELD selects the RSU from which byte 3 is to be tested. Bits 08 and 07 of the J FIELD select the Register Bit Pair to be compared against the Mask (bits 06 and 05 of the J FIELD). Refer to "Condition Selector" in this chapter for an explanation of Bit Pair Selector and Mask, as well as for the Test Bits.

This instruction tests for ones in the Test Bits. A logical one satisfies the test. The skip is executed by voiding the next instruction in the pipeline.

**NUMBER OF CYCLES:** SRB30 is a single-cycle instruction, if the condition is not met.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** SRB30 must only be used to skip single-cycle instructions.

**SRE****SKIP ON REGISTERS EQUAL****SRE****MNEMONIC:** SRE**OP CODE:** C3**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1 1 0 0 0 0 1 1								Source RSU		Source RSU Byte		Source RSU		Source RSU Byte	
OP Code; G, H, & I Fields								J Field				K Field			

GIM2271

**SUMMARY:** If condition, skip next sequential instruction;  
Else, execute next sequential instruction.

**OPERATION:** The bytes of data addressed by the J and K FIELDS are accessed and binarily compared, if the two bytes are equal the next instruction is not executed. The contents of the registers remain unchanged.

**NUMBER OF CYCLES:** SRE is a single-cycle instruction, if the condition is not met.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** SRE must only be used to skip single-cycle instructions.

**SRU****SKIP ON REGISTERS UNEQUAL****SRU****MNEMONIC:** SRU**OP CODE:** C2**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	0	0	0	1	0	Source RSU	Source RSU Byte	Source RSU	Source RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2272

**SUMMARY:** If condition, skip next sequential instruction;  
Else, execute next sequential instruction.

**OPERATION:** The bytes of data addressed by the J and K **FIELDS** are accessed and binarily compared, if the two bytes are unequal the next instruction is not executed. The contents of the registers remain unchanged.

**NUMBER OF CYCLES:** SRU is a single-cycle instruction if the condition is not met.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** SRU must only be used to skip single-cycle instructions.

# SUDF SUBTRACT UNPACKED DECIMAL FIELD SUDF

**MNEMONIC:** SUDF

**OP CODE:** 49

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	1	0	0	1	0	0	1	1	0	0	1	1	0	1	1
	OP Code; G, H, & I Fields								MARS4 Data RSU J Field = 9				MARS5 Data RSU K Field = 11			

GIM2234

**SUMMARY:**  $(RJ - B) - (RK - B) + C \rightarrow R13$ ; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, the low-order digit from the MARS5 Data Register specified by the K FIELD and the MARS5 Byte Pointers is subtracted decimally from the low-order digit from the MARS4 Data Register specified by the J FIELD and the MARS4 Byte Pointers. The result is placed in the low-order digit in the MARS6 Data Register (RSU13) specified by the MARS6 Byte Pointers. The Hex value 0011 is loaded into the high-order digit as the ASCII zone character. Following the subtraction, the Byte Pointers are decremented by one, and if one of the Byte Pointers crossed the word boundary, then the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** SUDF is a conditional multi-cycle instruction (maximum of 4 cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I4 and I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding SUDF must not alter the Tally Register or the MARS Address Registers used by SUDF.

**SW****SUBTRACT WORD****SW****MNEMONIC:** SW**OP CODE:** 4B**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	1	0	0	1	0	1	1	Source Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2224

**SUMMARY:** (RJ) - (RK) → RJ

**OPERATION:** A word from the RSU specified by the K FIELD is subtracted binarily from a word from the RSU specified by the J FIELD. The result replaces the operand in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SW is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

# SWAR SHIFT WORD ARITHMETIC RIGHT SWAR

**MNEMONIC:** SWAR

**OP CODE:** A0

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	0	0	0	0	0	Source Register				Dest Register			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2195

**SUMMARY:** (RJ) SHIFTED RIGHT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted right one bit. The least significant bit (01) is transferred to I4 in the Indicator Array (Carry). The most significant bit (32), which is the sign bit, remains in the 32nd bit position as well as being shifted to the 31st bit position.

**NUMBER OF CYCLES:** SWAR is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** None



**SWC****SUBTRACT WORD WITH CARRY****SWC****MNEMONIC:** SWC**OP CODE:** 4D**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	0	1	1	0	1	Source Dest RSU				Source RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2223

**SUMMARY:**  $(RJ) - (RK) + C \rightarrow RJ$ 

**OPERATION:** A word from the RSU specified by the K FIELD is subtracted binarily from a word from the RSU specified by the J FIELD. The Carry Indicator (I4) is binarily added to the least significant bit of the adder. The result replaces the operand in the RSU specified by the J FIELD. A borrow as a result of the subtraction will set I4 OFF, no borrow will set I4 ON.

**NUMBER OF CYCLES:** SWC is a single-cycle instruction**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I1, I2, I3, I4, and I5.**PROGRAMMING CONVENTIONS:** None

# SWCL      SHIFT WORD CIRCULAR LEFT      SWCL

**MNEMONIC:** SWCL

**OP CODE:** A5

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	1	0	0	1	0	1	Source Register				Dest Register			
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2194

**SUMMARY:** (RJ) SHIFTED LEFT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted left one bit. The most significant bit (32) is shifted to the least significant bit position (1). The contents of the source register remain unchanged.

**NUMBER OF CYCLES:** SWCL is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# SWL SUBTRACT WORD WITH LITERAL SWL

**MNEMONIC:** SWL

**OP CODE:** AF

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	1	1	1	Source Dest RSU				Digit Literal			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2225

**SUMMARY:** (RJ) - K → RJ

**OPERATION:** A word from the RSU specified by the J FIELD is decremented by the digit literal in the K FIELD. The result replaces the word in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SWL is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicators I1, I2, I3, I4, and I5.

**PROGRAMMING CONVENTIONS:** None

**SWLL****SHIFT WORD LOGICAL LEFT****SWLL****MNEMONIC:** SWLL**OP CODE:** A1**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	0	0	0	0	1	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2190

**SUMMARY:** (RJ) SHIFTED LEFT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted left one bit. The least significant bit (01) becomes a zero. The most significant bit (32) is transferred to the Carry Indicator (I4) in the Indicator Array. The contents of the source register remain unchanged.

**NUMBER OF CYCLES:** SWLL is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.**PROGRAMMING CONVENTIONS:** None

# SWLLC SHIFT WORD LOGICAL LEFT WITH CARRY SWLLC

**MNEMONIC:** SWLLC

**OP CODE:** A2

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	1	0	0	0	1	0	Source Register				Dest Register			
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2191

**SUMMARY:** (RJ) SHIFTED LEFT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted left one bit. The least significant bit (01) is set according to the previous value of I4 in the Indicator Array. The most significant bit (32) is transferred to I4 in the Indicator Array (Carry). The contents of the source register remain unchanged.

**NUMBER OF CYCLES:** SWLLC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** None

**SWLNI**

**SUBTRACT WORD LITERAL**  
**NO INDICATOR CHANGE**

**SWLNI**

**MNEMONIC:** SWLNI

**OP CODE:** 9D

**FORMAT:**

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1 0 0 1 1 1 0 1	Source/Dest RSU	Digit Literal
OP Code; G, H, & I Fields	J Field	K Field

GIM2226

**SUMMARY:** (RJ) – K → RJ

**OPERATION:** A word from the RSU specified by the J FIELD is decremented by the digit literal in the K FIELD. The result replaces the word in the RSU specified by the J FIELD.

**NUMBER OF CYCLES:** SWLNI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

SWLR

SHIFT WORD LOGICAL RIGHT

SWLR

MNEMONIC: SWLR

OP CODE: A3

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	0	0	1	1	Source Register				Dest Register			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2192

SUMMARY: (RJ) SHIFTED RIGHT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted right one bit. The most significant bit (32) is set to a zero. The least significant bit (01) is transferred to I4 in the Indicator Array (Carry). The contents of the source register remain unchanged.

**NUMBER OF CYCLES:** SWLR is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** None

# SWLRC SHIFT WORD LOGICAL RIGHT WITH CARRY SWLRC

**MNEMONIC:** SWLRC

**OP CODE:** A4

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	0	0	1	0	0	Source Register				Dest Register			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2193A

**SUMMARY:** (RJ) SHIFTED RIGHT → RK

**OPERATION:** The contents of the source RSU (32 bits) are accessed and shifted right one bit. The most significant bit (32) is set according to the previous value of I4 in the Indicator Array. The least significant bit (1) is transferred to I4 in the Indicator Array (Carry). The contents of the source register remain unchanged. ■

**NUMBER OF CYCLES:** SWLRC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I4.

**PROGRAMMING CONVENTIONS:** None



**TB**

**TRANSFER BYTE**

**TB**

**MNEMONIC:** TB

**OP CODE:** CD

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	0	1	1	0	1	Source RSU	Source RSU Byte	Dest RSU	Dest RSU Byte				
OP Code; G, H, & I Fields								J Field			K Field				

GIM2153

**SUMMARY:** (RJ - B) → RK - B

**OPERATION:** A byte from the RSU specified by the J FIELD is transferred to the byte in the RSU specified by the K FIELD. The other three bytes in RSU-K are not affected.

**NUMBER OF CYCLES:** TB is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TBF****TRANSFER BYTE TO FIELD****TBF****MNEMONIC:** TBF**OP CODE:** BF**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	1	1	1	Source RSU	Source RSU Byte	1	1	0	1
OP Code; G, H, & I Fields								J Field		MARS6 Data RSU K Field = 13			

GIM2168

**SUMMARY:** (RJ - B) → RK - B

**OPERATION:** A byte from the RSU specified by the J FIELD is transferred to the byte in the MARS6 Data Register specified by the K FIELD and the MARS6 Byte Pointers. The corresponding MARS6 Write Tag is set. The Tally Register is not used, the Byte Pointers are not modified and the MARS6 Overflow Flag is not set.

**NUMBER OF CYCLES:** TBF is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.**PROGRAMMING CONVENTIONS:** None

TBFD

TRANSFER BYTE TO  
FIELD DECREMENT

TBFD

MNEMONIC: TBFD

OP CODE: BC

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	1	0	0	Source RSU	Source RSU Byte	1	1	0	1
OP Code; G, H, & I Fields								J Field		MARS6 Data RSU K Field = 13			

GIM2169

**SUMMARY:** (RJ - B) → RK - B; decrement Byte Pointers; decrement Tally Register; set M6OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a byte from the RSU specified by the J FIELD is transferred to the byte in the MARS6 Data Register specified by the K FIELD and the MARS6 Byte Pointers. The corresponding MARS6 Write Tag is set. Following the transfer, the Byte Pointers are decremented by one and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

If the Tally Register decrements to zero, the MARS6 Overflow Flag will be set if the Byte Pointers crossed the word boundary.

**NUMBER OF CYCLES:** TBFD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

# TBFDN

## TRANSFER BYTE TO FIELD DECREMENT NO TALLY CHANGE

# TBFDN

**MNEMONIC:** TBFDN

**OP CODE:** BE

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	1	1	0	Source RSU	Source RSU Byte	1	1	0	1
OP Code; G, H, & I Fields								J Field		MARS6 Data RSU K Field = 13			

GIM2170

**SUMMARY:** (RJ - B) → RK - B; decrement Byte Pointers; set M6OF if word boundary.

**OPERATION:** A byte from the RSU specified by the J FIELD is transferred to the byte in the MARS6 Data Register specified by the K FIELD and the MARS6 Byte Pointers. The corresponding MARS6 Write Tag is set. Following the transfer, the Byte Pointers are decremented by one and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Direction Indicator Bit in the Field Array is set to a one. This instruction does not affect the Tally Register.

**NUMBER OF CYCLES:** TBFDN is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

**TBFI**

**TRANSFER BYTE TO  
FIELD INCREMENT**

**TBFI**

**MNEMONIC:** TBFI

**OP CODE:** BD

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	1	0	1	Source RSU	Source RSU Byte	1	1	0	1
OP Code; G, H, & I Fields								J Field		MARS6 Data RSU K Field = 13			

GIM2173

**SUMMARY:** (RK - B) → RJ - B; increment Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a byte from the RSU specified by the J FIELD is transferred to the byte in the MARS6 Data Register specified by the K FIELD and the MARS6 Byte Pointers. The corresponding MARS6 Write Tag is set. Following the transfer, the Byte Pointers are incremented by one and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a zero.

If the Tally Register decrements to zero, the MARS6 Overflow Flag will be set if the Byte Pointers crossed the word boundary.

**NUMBER OF CYCLES:** TBFI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

# TRANSFER BYTE TO FIELD INCREMENT NO TALLY CHANGE

**TBFIN**
**TBFIN**

**MNEMONIC:** TBFIN

**OP CODE:** BB

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	1	1	0	1	1	Source RSU	Source RSU Byte	1	1	0	1
OP Code; G, H, & I Fields								J Field		MARS6 Data RSU K Field = 13			

GIM2171

**SUMMARY:** (RJ - B) → RK - B; increment Byte Pointers; set M6OF if word boundary.

**OPERATION:** A byte from the RSU specified by the J FIELD is transferred to the byte in the MARS6 Data Register specified by the K FIELD and the MARS6 Byte Pointers. The corresponding MARS6 Write Tag is set. Following the transfer, the Byte Pointers are incremented by one and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Direction Indicator Bit in the Field Array is set to a zero. This instruction does not affect the Tally Register.

**NUMBER OF CYCLES:** TBFIN is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

TFB

TRANSFER FIELD TO BYTE

TFB

MNEMONIC: TFB

OP CODE: 2B

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	0	1	0	1	1	Dest RSU	Dest RSU Byte	MARS Data RSU					
OP Code; G, H, & I Fields								J Field		K Field = 9, 11 or 15					

GIM2177

SUMMARY: (RK - B) → RJ - B

OPERATION: A byte from MARS# Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the byte in the RSU specified by the J FIELD. The Tally Register is not used, the Byte Pointers are not modified and the MARS# Overflow Flag is not set.

NUMBER OF CYCLES: TFB is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I5 and I6.

PROGRAMMING CONVENTIONS: None

**TFBD****TRANSFER FIELD TO  
BYTE DECREMENT****TFBD**

MNEMONIC: TFBD

OP CODE: 28

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	0	1	0	0	0	Dest RSU	Dest RSU Byte	MARS Data RSU
OP Code; G, H, & I Fields								J Field		K Field = 9, 11, or 15

GIM2174

**SUMMARY:** (RK - B) → RJ - B; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a byte from MARS# Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the byte in the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are decremented by one and the MARS# Overflow Flag (M#OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** TBBD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None



TFBDN

TRANSFER FIELD TO  
BYTE DECREMENT  
NO TALLY CHANGE

TFBDN

MNEMONIC: TFBDN

OP CODE: 2A

FORMAT:

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

0	0	1	0	1	0	1	0	Dest RSU	Dest RSU Byte	MARS Data RSU
OP Code; G, H, & I Fields								J Field		K Field = 9, 11, or 15

GIM2175

SUMMARY: (RK - B) → RJ - B; decrement Byte Pointers; set M#OF if word boundary.

OPERATION: A byte from MARS# Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the byte in the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are decremented by one and the MARS# Overflow Flag (M#OF) is set if the Byte Pointers crossed the word boundary. The Direction Indicator Bit in the Field Array is set to a one. This instruction does not affect the Tally Register.

NUMBER OF CYCLES: TFBDN is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: TFBDN affects Indicator I5 and I6.

PROGRAMMING CONVENTIONS: None

**TFBI****TRANSFER FIELD TO  
BYTE INCREMENT****TFBI****MNEMONIC:** TFBI**OP CODE:** 29

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	0	1	0	0	1	Dest RSU		Dest RSU Byte		MARS Data RSU			
OP Code; G, H, & I Field								J Field			K Field = 9, 11, or 15				

GIM2176

**SUMMARY:** (RJ - B) → RK - B; increment Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a byte from MARS# Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the byte in the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are incremented by one and the MARS# Overflow Flag (M#OF) set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a zero.

**NUMBER OF CYCLES:** TFBI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

TFBIN

TRANSFER FIELD TO  
BYTE INCREMENT  
NO TALLY CHANGE

TFBIN

MNEMONIC: TFBIN

OP CODE: 3F

FORMAT: 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	1	1	1	1	1	Dest RSU	Dest RSU Byte	MARS Data RSU
OP Code; G, H, & I Fields								J Field		K Field = 9, 11 or 15

GIM2172

**SUMMARY:** (RK - B) → RJ - B; increment Byte Pointers; set M#OF if word boundary.

**OPERATION:** A byte from the MARS# Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the byte in the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are incremented by one and the MARS# Overflow Flag (M#OF) is set if the Byte Pointers crossed the word boundary. The Direction Indicator Bit in the Field Array is set to a zero. This instruction does not affect the Tally Register.

**NUMBER OF CYCLES:** TBFIN is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

**TFFD****TRANSFER FIELD TO  
FIELD DECREMENT****TFFD****MNEMONIC:** TFFD**OP CODE:** 26

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	0	0	1	1	0	1	1	0	1	1	0	0	1
	OP Code; G, H, & I Fields								MARS6 Data RSU J Field = 13				MARS4 Data RSU K Field = 9			

GIM2158

**SUMMARY:** (RK - B) → RJ - B; decrement Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from MARS4 Data Register (RSU9) specified by the K FIELD and the MARS4 Byte Pointers are transferred to the byte in the MARS6 Data Register (RSU13) specified by the J FIELD and the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are decremented by one and if either set of Byte Pointers crossed the word boundary, then the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** TFFD is a conditional multi-cycle instruction (maximum of four cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMINMG CONVENTIONS:** The instruction immediately preceding TFFD must not alter the Tally Register or the MARS Address Registers used by TFFD.

**TFFI****TRANSFER FIELD TO  
FIELD INCREMENT****TFFI****MNEMONIC:** TFFI**OP CODE:** 27**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	0	0	1	1	1	1	1	0	1	1	0	0	1
OP Code; G, H, & I Fields								MARS6 Data RSU J Field = 13				MARS4 Data RSU K Field = 9			

GIM2159

**SUMMARY:** (RK - B) → RJ - B; increment Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register equals zero, the execution of this instruction is voided. Otherwise, a byte from MARS4 Data Register (RSU9) specified by the K FIELD and the MARS4 Byte Pointers are transferred to the byte in the MARS6 Data Register (RSU13) specified by the J FIELD and the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are incremented by one and if either set of Byte Pointers crossed the word boundary, then the corresponding MARS Overflow Flag (M#OF) will be set. The Tally Register is decremented by one and the Direction Indicator Bit in the Field Array is set to a zero.

**NUMBER OF CYCLES:** TFFI is a conditional multi-cycle instruction (maximum of four cycles) that holds in the execution stage of the pipeline until a MARS overflow occurs or the Tally Register equals zero.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** The instruction immediately preceding TFFI must not alter the Tally Register or the MARS Address Registers used by TFFI.

TFLHD

TRANSFER FIELD TO  
LEFT HALFWORD DECREMENT

TFLHD

MNEMONIC: TFLHD

OP CODE: 22

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	0	0	0	1	0	Dest RSU				MARS Data RSU			
OP Code; G, H, & I Fields								J Field				K Field = 9, 11 or 15			

GIM2164

SUMMARY: (RK - H) → RJ - LH; decrement MARS Byte Pointers; decrement Tally Register; set M#OF if word boundary.

OPERATION: If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a halfword from the MARS Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the left halfword of the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are decremented by two and the MARS Overflow Flag (M#OF) set if the Byte Pointers crossed the word boundary. The Tally is decremented by two and the Direction Indicator Bit in the Field Array is set to a one.

NUMBER OF CYCLES: TFLHD is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

TFLHI

TRANSFER FIELD TO LEFT  
HALFWORD INCREMENT

TFLHI

MNEMONIC: TFLHI

OP CODE: 23

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	0	0	0	1	1	Dest RSU			MARS Data RSU				
	OP Code; G, H, & I Fields								J Field			K Field = 9, 11 or 15				

GIM2165

**SUMMARY:** (RK - H) → RJ - LH; increment MARS Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a halfword from the MARS Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the left halfword of the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are incremented by two and the Mars Overflow Flag (M#OF) set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a zero.

**NUMBER OF CYCLES:** TFLHI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TFRHD****TRANSFER FIELD TO RIGHT  
HALFWORD DECREMENT****TFRHD****MNEMONIC:** TFRHD**OP CODE:** 24**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	0	1	0	0	1	0	0	Dest RSU				MARS Data RSU			
OP Code; G, H, & I Fields								J Field				K Field = 9, 11 or 15			

GIM2166

**SUMMARY:** (RK - H) → RJ - RH; decrement MARS Byte Pointers; decrement Tally Register; set M#OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a halfword from the MARS Data Register specified by the K FIELD and the MARS# Byte Pointers is transferred to the right halfword of the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are decremented by two and the MARS Overflow Flag (M#OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a one.

**NUMBER OF CYCLES:** TFRHD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



TFRHI

TRANSFER FIELD TO RIGHT  
HALFWORD INCREMENT

TFRHI

MNEMONIC: TFRHI

OP CODE: 25

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	0	0	1	0	1	Dest RSU				MARS Data RSU			
OP Code; G, H, & I Fields								J Field				K Field = 9, 11 or 15			

GIM2167

SUMMARY: (RK - H) → RJ - RH; increment MARS Byte Pointers; decrement Tally Register; set M#OF if word boundary.

OPERATION: If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, a halfword from the MARS Data register specified by the K FIELD and the MARS# Byte Pointers is transferred to the right halfword of the RSU specified by the J FIELD. Following the transfer, the Byte Pointers are incremented by two and the MARS Overflow Flag (M#OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a zero.

NUMBER OF CYCLES: TFRHI is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**TIE**

**TRANSFER IN EXTERNAL (32-63)**

**TIE**

**MNEMONIC:** TIE

**OP CODE:** 00-01

<b>FORMAT:</b>	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	0	0	0	0	0	Register Addr Vector					Dest RSU			
	OP Code; G & H Fields							I & J Fields					K Field			

GIM2180

**SUMMARY:** (XIJ) → RK

**OPERATION:** The contents of one of the 64 Processor Registers specified by the I,J FIELDS are transferred into the RSU specified by the K FIELD. The Register Address Vector is a binary value which is added to a base value of 32<sub>10</sub>. This gives the decimal ERU number. In the case where the register is less than 32 bits, and is one of the 32 ERUs, all unspecified bits will be set to zero in the RSU.

**NUMBER OF CYCLES:** TIE is a conditional multi-cycle instruction due to contention on the PM Bus.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

TII

TRANSFER IN INTERNAL (0-31)

TII

MNEMONIC: TII

OP CODE: 20-21

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	0	0	1	0	0	0	0	Register Addr Vector					Dest RSU			
	OP Code; G & H Fields							I & J Fields					K Field			

GIM2181

SUMMARY: (XIJ) → RK

**OPERATION:** The contents of one of the 64 Processor Registers specified by the I, J FIELDS are transferred into the RSU specified by the K FIELD. In the case where the register is one of the 32 IRUs, it will be transferred either right-justified or left-justified as a 16 bit container. The other 16 bits of the destination RSU will not be disturbed. If the IRU is less than 16 bits in length then the unspecified bits will be set to zero.

**NUMBER OF CYCLES:** TII is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TIP****TRANSFER IN PORT (64-127)****TIP****MNEMONIC:** TIP**OP CODE:** 0C-0F**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	1	1	I/O Port Addr					Dest RSU				
OP Code; G & H Fields						I & J Fields					K Field				

GIM2183

**SUMMARY:** (XIJ) → RK

**OPERATION:** The contents of the I/O Port (one of the last 64 ERUs) on the PM Bus specified by the I,J FIELDS are transferred into the RSU specified by the K FIELD. The I/O Port Address is a binary value which is added to a base value of  $64_{10}$ . This gives the decimal ERU number. In cases where the I/O Port is less than 32 bits, all unspecified bits must be set to zero by the I/O Port.

**NUMBER OF CYCLES:** TIP is a single-cycle instruction, although contention on the PM Bus may halt the Processor until the PM Bus is available.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**TLDLD**

**TRANSFER LEFT DIGIT  
TO LEFT DIGIT**

**TLDLD**

**MNEMONIC:** TLDLD

**OP CODE:** D2

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	1	0	1	0	0	1	0	Source RSU	Source RSU Byte	Dest RSU	Dest RSU Byte
OP Code; G, H, & I Fields								J Field		K Field	

GIM2154

**SUMMARY:** (RJ - LD) → RK - LD

**OPERATION:** The left digit in the byte specified by the J FIELD is transferred to the left digit in the byte of the RSU specified by the K FIELD. The right digit in the destination byte is not affected.

**NUMBER OF CYCLES:** TLDLD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

**TLDRD**

**TRANSFER LEFT DIGIT  
TO RIGHT DIGIT**

**TLDRD**

**MNEMONIC:** TLDRD

**OP CODE:** D3

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	0	0	1	1	Source RSU		Source RSU Byte		Dest RSU		Dest RSU Byte	
OP Code; G, H, & I Fields								J Field			K Field				

GIM2155

**SUMMARY:** (RJ – LD) → RK – RD

**OPERATION:** The left digit in the byte specified by the J FIELD is transferred to the right digit in the byte of the RSU specified by the K FIELD. The left digit in the destination byte is not affected.

**NUMBER OF CYCLES:** TLDRD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

**TLHFD****TRANSFER LEFT HALFWORD  
TO FIELD DECREMENT****TLHFD****MNEMONIC:** TLHFD**OP CODE:** A6**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	0	0	1	1	0	Source RSU				1	1	0	1
OP Code; G, H, & I Fields								J Field				MARS6 Data RSU K Field = 13			

GIM2160

**SUMMARY:** (RJ - LH) → RK - H; decrement MARS6 Byte Pointers; decrement Tally Register; set M6OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, the left halfword of the RSU specified by the J FIELD is transferred to RSU13 (MARS6 Data Register) specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are decremented by two and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a one.

If the Tally Register decrements to zero and one or more of the MARS6 Write Tags are on, the MARS6 Overflow Flag will be set.

**NUMBER OF CYCLES:** TLHFD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TLHFI****TRANSFER LEFT HALFWORD  
TO FIELD INCREMENT****TLHFI****MNEMONIC:** TLHFI**OP CODE:** A7**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	0	1	1	1	Source RSU				1	1	0	1
OP Code; G, H, & I Fields								J Field				MARS6 Data RSU K Field = 13			

GIM2161

**SUMMARY:** (RJ - LH) → RK - H; increment MARS6 Byte Pointers; decrement Tally Register; set M6OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, the left halfword of the RSU specified by the J FIELD is transferred to RSU13 (MARS6 Data Register) specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are incremented by two and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a zero.

If the Tally Register decrements to zero and one or more of the MARS6 Write Tags are on, the MARS6 Overflow Flag will be set.

**NUMBER OF CYCLES:** TLHFI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



TLHLH

TRANSFER LEFT HALFWORD  
TO LEFT HALFWORD

TLHLH

MNEMONIC: TLHLH

OP CODE: AA

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	1	0	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2149A

SUMMARY: (RJ - LH) → RK - LH

OPERATION: The left halfword in the RSU specified by the J FIELD is transferred to the left halfword in the RSU specified by the K FIELD. The right halfword in the destination RSU is not affected.

NUMBER OF CYCLES: TLHLH is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**TLHRH**

**TRANSFER LEFT HALFWORD  
TO RIGHT HALFWORD**

**TLHRH**

**MNEMONIC:** TLHRH

**OP CODE:** AB

**FORMAT:**

16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	1	0	1	0	1	1	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2150A

**SUMMARY:** (RJ - LH) → RK - RH

**OPERATION:** The left halfword in the RSU specified by the J FIELD is transferred to the right halfword in the RSU specified by the K FIELD. The left halfword in the destination RSU is not affected.

**NUMBER OF CYCLES:** TLHRH is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

TOE

TRANSFER OUT EXTERNAL (32-63)

TOE

MNEMONIC: TOE

OP CODE: 10-11

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	0	0	Register Addr Vector					Source RSU			
OP Code; G & H Fields							I & J Fields					K Field			

GIM2178

SUMMARY: (RK) → XIJ

**OPERATION:** The contents of the RSU specified by the K FIELD are transferred into one of the 32 ERUs specified by the I,J FIELDS and portions of the OP-CODE. The Register Address Vector is a binary value which is added to a base value of 32<sub>10</sub>. This gives the decimal ERU number.

**NUMBER OF CYCLES:** TOE is a conditional multi-cycle instruction due to contention on the PM Bus.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TOI****TRANSFER OUT INTERNAL (0-31)****TOI****MNEMONIC:** TOI**OP CODE:** 30-31**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	1	1	0	0	0	Register Addr Vector					Source RSU			
OP Code; G & H Fields							I & J Fields					K Field			

GIM2179

**SUMMARY:** (RK) → XIJ

**OPERATION:** The contents of the RSU specified by the K FIELD are transferred into one of the 32 IRUs specified by the I, J, FIELDS and a portion of the OP-CODE.

**NUMBER OF CYCLES:** TOI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**TOP****TRANSFER OUT PORT (64-127)****TOP****MNEMONIC:** TOP**OP CODE:** 1C-1F**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	1	1	I/O Port Addr					Source RSU				
OP Code; G & H Fields						I & J Fields					K Field				

GIM2182

**SUMMARY:** (RK) → XIJ

**OPERATION:** The contents of the RSU specified by the K FIELD are transferred to the I/O Port (one of the last 64 ERUs) on the PM Bus. The I/O Port Address is a binary value which is added to a base value of  $64_{10}$ . This number gives the decimal ERU number.

**NUMBER OF CYCLES:** TOP is a single-cycle instruction, although contention on the PM Bus may halt the processor until the PM Bus is available.

**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**TRDLD****TRANSFER RIGHT DIGIT  
TO LEFT DIGIT****TRDLD****MNEMONIC:** TRDLD**OP CODE:** D4

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1 1 0 1 0 1 0 0								Source RSU		Source RSU Byte		Dest RSU		Dest RSU Byte	
OP Code; G, H, & I Fields								J Field				K Field			

GIM2156

**SUMMARY:** (RJ - RD) → RK - LD

**OPERATION:** The right digit in the byte specified by the J FIELD is transferred to the left digit in the byte of the RSU specified by the K FIELD. The right digit in the destination byte is not affected.

**NUMBER OF CYCLES:** TRDLD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** This instruction affects Indicator I5 and I6.

**PROGRAMMING CONVENTIONS:** None

TRDRD

TRANSFER RIGHT DIGIT  
TO RIGHT DIGIT

TRDRD

MNEMONIC: TRDRD

OP CODE: D5

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	0	1	0	1	Source RSU	Source RSU Byte	Dest RSU	Dest RSU Byte				
OP Code; G, H, & I Fields								J Field			K Field				

GIM2157

SUMMARY: (RJ – RD) → RK – RD

OPERATION: The right digit in the byte specified by the J FIELD is transferred to the right digit in the byte specified by the K FIELD. The left digit in the destination byte is not affected.

NUMBER OF CYCLES: TRDRD is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: This instruction affects Indicator I5 and I6.

PROGRAMMING CONVENTIONS: None

# TRHFD      TRANSFER RIGHT HALFWORD TO FIELD DECREMENT      TRHFD

MNEMONIC: TRHFD

OP CODE: A8

FORMAT:	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	0	1	0	1	0	0	0	Source RSU				1	1	0	1
	OP Code; G, H, & I Fields								J Field				MARS6 Data RSU K Field = 13			

GIM2162

**SUMMARY:** (RJ - RH) → RK - H; decrement MARS6 Byte Pointers; decrement Tally Register; set M6OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, the right halfword of the RSU specified by the J FIELD is transferred to RSU13 (MARS6 Data Register) specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are decremented by two and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a one.

If the Tally Register decrements to zero and one or more of the MARS6 Write Tags are on, the MARS6 Overflow Flag will be set.

**NUMBER OF CYCLES:** TRHFD is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



# TRHFI TRANSFER RIGHT HALFWORD TO FIELD INCREMENT TRHFI

MNEMONIC: TRHFI

OP CODE: A9

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	0	0	1	Source RSU				1	1	0	1
OP Code; G, H, & I Fields								J Field				MARS6 Data RSU K Field = 13			

GIM2163

**SUMMARY:** (RJ - RH) → RK - H; increment MARS6 Byte Pointers; decrement Tally Register; set M6OF if word boundary.

**OPERATION:** If the Tally Register is equal to zero, the execution of this instruction is voided. Otherwise, the right halfword of the RSU specified by the J FIELD is transferred to RSU13 (MARS6 Data Register) specified by the MARS6 Byte Pointers. Following the transfer, the Byte Pointers are incremented by two and the MARS6 Overflow Flag (M6OF) is set if the Byte Pointers crossed the word boundary. The Tally Register is decremented by two and the Direction Indicator Bit in the Field Array is set to a zero.

If the Tally Register decrements to zero and one or more of the MARS6 Write Tags are on, the MARS6 Overflow Flag will be set.

**NUMBER OF CYCLES:** TRHFI is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# TRHLH      TRANSFER RIGHT HALFWORD TO LEFT HALFWORD      TRHLH

**MNEMONIC:** TRHLH

**OP CODE:** AC

**FORMAT:**      16 15 14 13 12 11 10 9    8 7 6 5 4 3 2 1

1	0	1	0	1	1	0	0	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2151

**SUMMARY:** (RJ - RH) → RK - LH

**OPERATION:** The right halfword in the RSU specified by the J FIELD is transferred to the left halfword in the RSU specified by the K FIELD. The right halfword in the destination RSU is not affected.

**NUMBER OF CYCLES:** TRHLH is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

TRHRH

TRANSFER RIGHT HALFWORD  
TO RIGHT HALFWORD

TRHRH

MNEMONIC: TRHRH

OP CODE: AD

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	1	0	1	1	0	1	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2152

SUMMARY: (RJ-RH) → RK - RH

OPERATION: The right halfword in the RSU specified by the J FIELD is transferred to the right halfword in the RSU specified by the K FIELD. The left halfword in the destination RSU is not affected.

NUMBER OF CYCLES: TRHRH is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**TSB****TRANSFER BYTE FROM SETUP****TSB****MNEMONIC:** TSB**OP CODE:** 96**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	0	1	1	0	Dest RSU				0	0	0	0
OP Code; G, H, & I Fields								J Field				K Field; Not Used			

GIM2302

**SUMMARY:** (SUR1 08 - 01) → RJ

**OPERATION:** Bits 08-01 of SUR1 (the additional byte of the Virtual Op Code for the S Format or the I(2) literal for the SI Format during IBM emulation) are transferred right-justified and zero filled in byte 2 to the right-half of the RSU specified by the J FIELD. The left-half is not disturbed. During VRX emulation, the byte transferred corresponds to either the Ra or the Rb field.

This instruction is also valid for any Virtual Machine emulation where SUR1 is used.

**NUMBER OF CYCLES:** TSB is a single-cycle instruction.**EFFECT ON INDICATOR ARRAY:** None**PROGRAMMING CONVENTIONS:** None

**TSBC**

**TRANSFER BYTE FROM  
SETUP AND CLEAR**

**TSBC**

**MNEMONIC:** TSBC

**OP CODE:** 99

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

1	0	0	1	1	0	0	1	Dest RSU				0	0	0	0
OP Code; G, H, & I Fields								J Field				K Field; Not Used			

GIM2277

**SUMMARY:** (SUR1 08 - 01) → RJ

**OPERATION:** Bits 08-01 of SUR1 (the additional byte of the Virtual Op Code for the S Format or the I(2) literal for the SI Format during IBM evaluation) are transferred right-justified and zero filled to the RSU specified by the J FIELD. During VRX emulation the byte transferred corresponds to either the Ra or Rb field.

This instruction is also valid for any Virtual Machine emulation where SUR1 is used.

**NUMBER OF CYCLES:** TSBC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

# TSLDC      TRANSFER LEFT DIGIT FROM      TSLDC

## SETUP AND CLEAR

**MNEMONIC:** TSLDC

**OP CODE:** 97

**FORMAT:**      16 15 14 13 12 11 10 9      8 7 6 5 4 3 2 1

1	0	0	1	0	1	1	1	Dest RSU	0	0	0	0
OP Code; G, H, & I Fields								J Field	K Field; Not Used			

GIM2285

**SUMMARY:** (SUR1 08 – 05) → RJ

**OPERATION:** Bits 08-05 of SUR1 (the literal R1 or N of the Virtual Instruction during NVM emulation) are transferred right-justified and zero filled to the RSU specified by the J FIELD.

This instruction is also valid for any Virtual Machine emulation where SUR1 is used.

**NUMBER OF CYCLES:** TSLDC is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

TSRDC

TRANSFER RIGHT DIGIT FROM  
SETUP AND CLEAR

TSRDC

MNEMONIC:    TSRDC

OP CODE:       98

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	1	1	0	0	0	Dest RSU				0	0	0	0
OP Code; G, H, & I Fields								J Field				K Field; Not Used			

GIM2286

SUMMARY:     (SUR1 04 – 01) → RJ

OPERATION:   Bits 04-01 of SUR1 (the literal R2 or M of the Virtual Instruction during NVM emulation) are transferred right-justified and zero filled to the RSU specified by the J FIELD.

          This instruction is also valid for any Virtual Machine emulation where SUR1 is used.

NUMBER OF CYCLES:  TSRDC is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY:   None

PROGRAMMING CONVENTIONS:  None

**TW**

**TRANSFER WORD**

**TW**

**MNEMONIC:** TW

**OP CODE:** 51

**FORMAT:** 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

0	1	0	1	0	0	0	1	Source RSU				Dest RSU			
OP Code; G, H, & I Fields								J Field				K Field			

GIM2148

**SUMMARY:** (RJ) → RK

**OPERATION:** A word (32 bits) from the RSU specified by the J FIELD is transferred to the word in the RSU specified by the K FIELD.

**NUMBER OF CYCLES:** TW is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None



UPKL

UNPACK LEFT DIGIT

UPKL

MNEMONIC: UPKL

OP CODE: D6

FORMAT:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	0	1	0	1	1	0	Source RSU	Source RSU Byte	Dest RSU	Dest RSU Byte				
OP Code; G, H, & I Fields								J Field			K Field				

GIM2287

SUMMARY: (RJ - BLD) → RK - BRD  
          ZONE → RK - BLD

OPERATION: The left digit in the byte specified by the J FIELD is transferred to the right digit in the byte of the RSU specified by the K FIELD. The value of the transferred digit is monitored. If it is equal to nine or less, the ASCII zone character 3 is inserted into the left digit of the destination byte. If it is greater than nine, the ASCII zone character 2 is inserted into the left digit of the destination byte.

NUMBER OF CYCLES: UPKL is a single-cycle instruction.

EFFECT ON INDICATOR ARRAY: None

PROGRAMMING CONVENTIONS: None

**UPKR****UNPACK RIGHT DIGIT****UPKR****MNEMONIC:** UPKR**OP CODE:** D7

**FORMAT:**

	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
	1	1	0	1	0	1	1	1	Source RSU		Source RSU Byte		Dest RSU		Dest RSU Byte	
	OP Code; G, H, & I Fields								J Field				K Field			

GIM2288

**SUMMARY:** (RJ - BRD) → RK - BRD  
 ZONE → RK - BLD

**OPERATION:** The right digit in the byte specified by the J FIELD is transferred to the right digit in the byte of the RSU specified by the K FIELD. The value of the digit is monitored. If it is equal to nine or less, the ASCII Zone character 3 is inserted into the left digit of the destination byte. If it is greater than nine, the ASCII Zone character 2 is inserted into the left digit of the destination byte.

**NUMBER OF CYCLES:** UPKR is a single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**WPMB**

WAIT ON PMBUS

**WPMB**

**MNEMONIC:** WPMB

**OP CODE:** 13

**FORMAT:**

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0
OP Code; G, H, & I Fields								J Field; Not Used				K Field; Not Used			

GIM2275

**SUMMARY:** No operation

**OPERATION:** This instruction performs no operation but does require PM Bus availability. If the PM Bus is available then WPMB executes in a single cycle. Otherwise, the Processor stops until the Bus becomes available before executing the instruction.

**NUMBER OF CYCLES:** WPMB is a conditional single-cycle instruction.

**EFFECT ON INDICATOR ARRAY:** None

**PROGRAMMING CONVENTIONS:** None

**APPENDICES****CONTENTS**

Appendix A, Glossary of Terms. . . . .	A-1
Appendix B, Setup Flows. . . . .	B-1
Appendix C, Breakpoint Operation. . . . .	C-1
Appendix D, Memory Retries. . . . .	D-1
Appendix E, Fetching from ISU. . . . .	E-1
Appendix F, Non-Interruptable Instructions. . . . .	F-1
Appendix G, Array Matrices. . . . .	G-1
Appendix H, PBCD and UBCD Setting. . . . .	H-1
Appendix I, Instruction Emulation Example. . . . .	I-1

## APPENDIX A

### GLOSSARY OF TERMS

TERM	DEFINITION/DESCRIPTION
ALU	<p>Arithmetic Logic Unit:</p> <p>The unit which performs binary and decimal arithmetic operations, boolean operations, and shift functions in the CPC.</p>
AM	<p>Associative Memory:</p> <p>There are 16 AM entries in the DAT unit. Each AM entry contains a 22-bit Virtual Page Number (VPN).</p>
AMR	<p>Address Monitor Register:</p> <p>A 32-bit double stage register in the ATC used to monitor virtual addresses during Virtual Memory Message transfers.</p>
AT	<p>Address Translation:</p> <p>The flag located in Control Array #1 which controls whether the ATC translates virtual messages into real messages.</p>
ATC	Address Translation Chip (NCR/32-010)
BAC	Bus Assist Chip (NCR/32-801)
bank	A block of memory defined by the number of addresses and the number of bits at each address.
BAV	<p>Bus Available:</p> <p>A signal asserted by the Bus Priority Logic to indicate to the CPC that the PM Bus will be available on the next System Clock cycle.</p>

TERM	DEFINITION/DESCRIPTION
BCT	Between Commands Testing: A firmware flag located in Control Array #1 of the CPC and used to branch to special routines.
BIN	Bus Interrupt Register: A 32-bit register in the ATC used by System Interface Chips (SIC) to report input/output status.
bit	A single binary logic data unit.
Bus Priority Logic	The circuitry that controls the time-sharing of the PM Bus. Basically, it monitors the <u>REQ</u> lines and sets BAV and the <u>SEL</u> lines to enable use of the PM Bus.
byte	8 bits of data in parallel or series.
BWE	Byte Write Enable: Bits 28-25 of the PM Bus on Real Memory Stores which act as "Write Enables" for Bytes 0 to 3.
CAS	Column Address Signal: A signal set by the Memory Interface to clock the 8-bit column address (PMBUS11-18) into the 64K Dynamic RAM Address Latch.
CLOCK0	The first half, or Phase 0, of a System Clock cycle. Also referred to in text as "X0".
CLOCK1	The second half, or Phase 1 of a System Clock cycle. Also referred to in text as "X1".
CMD Register	Command Register: A 32-bit register in the SIC loaded by the CPC with command information using a TOE operation. When read with a TIE the CPC will retrieve the SIC status.

TERM	DEFINITION/DESCRIPTION
Control Array #1	An 8-bit register in the CPC which holds control bits and flags used during instruction executions.
Control Array #2	A 16-bit register in the ATC used to set the operating mode of the ATC. It is accessible with a TIE or TOE.
CPC	Central Processor Chip (NCR/32-000)
CR	Control Register: A 16-bit counting register in the CPC used to address the instructions in the ISU ROM.
DAT	Dynamic Address Translation: A unit in the ATC used during Virtual Memory Fetch and Store operations to translate virtual addresses to real addresses.
DAT No Match Interrupt	Dynamic Address Translation No Match Interrupt: An interrupt generated by the DAT unit in the ATC when an address search of the AM fails to find a matching address.
DDR	Descriptor Data Register: A 32-bit register in the ATC used to hold the contents of the RAR the cycle following a Virtual Memory operation. The DDR acts as a history register for the RAR and may be accessed with a TIE.
$\overline{\text{DIE}}$	Data Input Enable: A Signal set at X1 and held through the next X0 by the Memory Interface to indicate to the CPC or ATC that data for a Memory Fetch operation will be available during the next System Clock cycle. During

TERM	DEFINITION/DESCRIPTION
	Memory Store operations the Memory Interface asserts DIE to indicate it is ready to accept data.
EAC	Extended Arithmetic Chip (NCR/32-020)
<u>EACI</u>	Extended Arithmetic Chip Information: A signal output by the EAC indicating that the result of a math or logic operation is ready for retrieval.
ECC	Error Correction/Check: Circuitry in the ATC that checks the accuracy of data on the PM Bus using the Syndrome (check) Bits. When possible the data is corrected. If a data error cannot be corrected an error signal is passed to the CPC or SIC to indicate that intervention is required.
<u>EREP</u>	External Register Enable/Permit: A signal set by the CPC or SIC at X0 to notify the ATC, EAC, or SIC that one of its External Registers is being addressed. The ATC will assert <u>EREP</u> at X1 when the ATC decodes a write to the BIN register.
ERU	External Register Unit: Any one of the registers external to the CPC available for fetch/store operations via the PM Bus.
Execute Stage	The third stage of the 3-stage CPC pipeline which executes instructions and writes results into appropriate RSUs.
Fetch Stage	The first stage of the 3-stage CPC pipeline which fetches instructions from the ISU ROM and writes them into the IR.
field	A block of memory defined by its length and start address (normally 1 to 64K-1 bytes).



TERM	DEFINITION/DESCRIPTION
Field Mask	(See Test Mask Operand and Indicator Array)
Full-Word	Same as a Word; 32-bits of data in parallel or in series.
Half-Word	Left and Right: 16 bits of data in parallel or in series.
I-Bus	The main internal 32-bit bus within the CPC which interconnects the RSU, the IR, the ALU, the CR, the restore FIFO and the PM Bus Interface.
I/O	Input/Output (normally an interface).
IAR	Instruction Address Register: A 16-bit register in the CPC which holds the address of the instruction in the Interpret Stage of the pipeline.
IMR	Interrupt Mask Register: A register in the ATC which provides selective masking of all interrupt bits in the I/TA.
Indicator Array	An 8-bit register (IRU16) in the CPC which contains indicator flags that reflect the results of an instruction execution. (See also Test Mask Operand.)
INH	Inhibit: An input line which may be used to disable the Refresh control in the ATC when either static RAM or an external refresh controller is used.
Instruction (micro)	Any one of the 179 Op-codes from the CPC's Microcode Instruction Set.

TERM	DEFINITION/DESCRIPTION
INT	<p><b>Interrupt:</b></p> <p>An input signal to the CPC indicating the presence of an interrupt condition.</p>
Interpret Stage	<p>The second stage of the 3-stage CPC pipeline which decodes instructions and reads operands from the RSU.</p>
IR	<p><b>Instruction Register:</b></p> <p>A 16-bit register in the CPC which holds the instructions fetched from the ISU ROM.</p>
IRU	<p><b>Internal Register Unit:</b></p> <p>One of 22 special-purpose registers in the CPC designed to ease virtual machine emulation.</p>
ISU	<p><b>Instruction Storage Unit:</b></p> <p>A functional unit consisting of ROM chips which contain the microcode instructions fetched and executed by the CPC.</p>
ISU01-16	<p><b>Instruction Storage Bus (bits 01 to 16):</b></p> <p>The bus between the CPC and the ISU used to fetch instructions:</p>
ISU ROM	<p>Instruction Storage Unit ROM (NCR/32-901)</p>
I/TA	<p><b>Interrupt/Trap Array:</b></p> <p>Each trap or interrupt has a unique bit assigned to it in the ATC I/TA.</p>
ITMR	<p><b>Interval Timer/Monitor Register:</b></p> <p>A 32-bit register in the ATC that contains a value corresponding to a desired interval of time. When the ITMR and TOD Registers match a TOD Interrupt is initiated to the I/TA.</p>

TERM	DEFINITION/DESCRIPTION
JRJ	<p>Jump Register "J":</p> <p>The jump register in the CPC addressed by the J-Field.</p>
Jump Registers	Eight addressable 16-bit registers in the CPC that can be used to hold jump addresses.
literal	The term applied to a binary code that represents a numeric value, an ASCII character, or a memory address in the Operand Field of the CPC Instruction Set.
<u>MAE</u>	<p>Memory Address Enable:</p> <p>A signal asserted by the CPC, ATC, or SIC at X0 when making a Real Memory Message transfer via the PM Bus.</p>
Main Memory	The primary storage area for programs and data. The Main Memory is connected to the PM Bus via the Memory Interface.
MARS	<p>Memory Assist Registers:</p> <p>Odd/even RSU pairs of the 16 RSU registers, numbered MARS0 to MARS7, and used during field instruction execution or memory store operations.</p>
MARS Byte Pointers	The two least significant bits of RSU8(MARS4), RSU10(MARS5), RSU12(MARS6), and RSU14(MARS7) used for indirectly addressing RSU9, RSU11, RSU13, and RSU15.
MARS Write Tag	A 4-bit code loaded into the Write Tag Register and output as Byte Write Enables (PMWT0-3) on the PM Bus.
<u>MDEE</u>	<p>Memory Data Enable/Error:</p> <p>A signal set by the Memory Interface at X0 to indicate that memory data is available for a Memory Fetch. The ATC will assert</p>

TERM	DEFINITION/DESCRIPTION
	<u>MDEE</u> at X1 if the data on the PM Bus contains an uncorrectable error.
<u>MEMERR</u>	Memory Error: A signal set at X1 by the ATC to indicate any error (single or multiple) condition detected by the ECC logic during Memory Fetch, Store, or Refresh operations.
message	Any combination of bits transferred during a Memory Fetch or Store operation.
microcode	A bit or group of bits assigned a specific function. May also be the same as a micro-instruction.
microcommand	The 8-bit Op-Code portion of a microinstruction from the CPC Instruction Set.
micro-instruction	A 16 or 32-bit instruction from the CPC Instruction Set.
MSU	Memory Storage Unit: The MSU is synonymous with Main Memory.
nibble	4 bits of data or half of a Byte.
NIE	Normal Interrupt Enable: The flag in CPC Control Array #1 which enables the recognition of interrupts.
NVM	New Virtual Machine: The Virtual Machine for all new NCR Software products and designed for the NCR 9300 series.
Op-Code	Operation Code: Has the same meaning as "Command Code" in this manual and refers to a portion (G, H,

TERM	DEFINITION/DESCRIPTION
	I Fields) of the microinstruction in the CPC Instruction Set.
Operand	Operands are 4, 8, 16, or 32-bit literals. They contain an address, binary/decimal data, a set of code flags, a printable ASCII character, or a combination of these.
Operand Pointer #1 & #2	These two 7-bit incrementing/decrementing pointers are used to access the Scratch Pad portion of memory.
Operand Registers	The J and K Operand Registers in the CPC which are loaded from the RSU during the Interpret Stage.
Overflow Flags	Indicators which are set in the CPC during Field Instructions or Setup Instructions when a MARS Byte Pointer crosses the word boundary.
PM Bus	Processor-Memory Bus (1 to 32): The common address/data bus used for transferring data (Messages) between the NCR/32 Processor Family system units. The bus includes several handshaking and control lines.
packed BCD	Data representation in which two Binary Coded Decimal digits are contained in one byte.
Page Descriptor	The Page Descriptor, used in the DAT unit of the ATC, contains 25-bits: 8-bits of Protection Check, a 14-bit Page Frame Number, 2-bits of AM entry control, and an Invalid Register (entry) bit.
PFN	Page Frame Number: A part of the Page Descriptor in the ATC which contains 12 to 14 bits. The PFN is

TERM	DEFINITION/DESCRIPTION
	concatenated with the page displacement to form the Real Memory address.
<u>PMCHK1-7</u>	Processor-Memory Check (1 to 7): The seven Syndrome (check) Bits sent or received by the Memory Interface at X0 and used by the ECC logic in the ATC to maintain integrity of the data on the PM Bus.
PMR	Purge Mask Register: A 22-bit register in the ATC that is used to selectively purge the Associative Memory (AM). When one or more bits are set in the PMR, the AM ignores the corresponding VAR bits in the association process for the Purge operation.
<u>PMRST</u>	Processor-Memory Reset: A common reset signal to all PM Bus interfaces during power-up or programmable reset sequences to initialize all appropriate logic.
<u>PMWT0-3</u>	Processor Memory Write Tags (0 to 3): Assigned to Bytes 0 through 3 as "Write Enables". The appropriate tags are set during Virtual Memory Store Message operations.
<u>PRIV</u>	Privilege: An ATC input pin used to directly set or reset the Privilege Bit in Control Array #2. It is reserved for use by a future performance booster chip. Assertion of PRIV during X0 allows clearing or setting of the Privilege Flag during the subsequent X1 clock according to the state of PRIV at that time; where assertion of PRIV during X1 sets the Privilege Flag.

TERM	DEFINITION/DESCRIPTION
Privilege Flag	See PRIV above.
PSR	<p>Page Size Register:</p> <p>A 3-bit register in the ATC which specifies the page size in use.</p>
$\overline{\text{PVT}}$	<p>Processor Virtual Transfer:</p> <p>A PM Bus signal asserted by the CPC during X0 (phase 0) when transferring a Virtual Memory Message, and during X1 on all PM Bus memory transfers.</p>
$\overline{\text{PWFAIL}}$ (SPINT)	<p>Power Failure (or Special Interrupt):</p> <p>A signal activated by the system Power Control Logic to notify the CPC that a power failure is imminent. It may also be used for any other special-purpose external interrupt condition.</p>
RAM	<p>Random Access Memory:</p> <p>May be Dynamic (DRAM) or Static (SRAM).</p>
RAR	<p>Real Address Register:</p> <p>A 32-bit register in the ATC that contains the Real Memory address and the Byte Write Tags. The ATC will load the RAR from the PM Bus during CPC or SIC Real Memory Fetch operations, and from the DAT during Virtual Memory operations.</p>
RAS	<p>Row Address Signal:</p> <p>A signal set by the Memory Interface to clock the 8-bit row address (PMBUS03-10) into the 64K Dynamic RAM Address Latch.</p>
Real Memory	<p>A memory access operation which applies the address directly via the PM Bus without address translation.</p>

TERM	DEFINITION/DESCRIPTION
<u>REQ0-n</u>	Subsystem Bus Request (Line 0 to n): Priority request lines, one for each unit on the PM Bus, used to indicate a need to use the PM Bus. The lines are active at X0. <u>REQ0</u> is the highest priority and always assigned to the ATC.
REQS	Request Special: A signal asserted by the ATC under special conditions to give the PM Bus to the CPC and block all other requests for access to the PM Bus.
reset	Used to indicate a signal in its inactive state. It may be either an inactive high (logic 1) or an inactive low (logic 0) depending on signal polarities. See also "set".
Restore FIFO	A three-deep, 16-bit, first in, first out (FIFO) shift register in the CPC used to hold the addresses of the instructions in the pipeline at the time of a trap or an interrupt.
ROM	Read Only Memory: Memory which can only be read, not written.
RSU	Register Storage Unit: The unit in the CPC which contains the sixteen 32-bit general purpose registers. See also MARS.
Scratch Pad	RAM area set aside for the CPC firmware to use as fast-access temporary storage addressable via the Operand Pointers.
<u>SEL1-n</u>	Select (1 to n): The signal lines asserted by the Bus Priority Logic circuit and sampled by the PM Bus devices at X1 to establish which device has access to the PM Bus during the next



TERM	DEFINITION/DESCRIPTION
	System Clock cycle. The highest priority is automatically the ATC; therefore SEL0 is not used.
set	Used to indicate a signal in its active state. It may be either an active high (logic 1) or active low (logic 0) depending on the signal polarities. See also "Reset".
Set Up Registers	See SUR1-5
SIC	System Interface Controller (NCR/32-500)
SIR	System Interface Receiver (NCR/32-590)
SIT	System Interface Transmitter (NCR/32-580)
<u>SPINT</u>	Special Interrupt: An ATC special-purpose interrupt input (see <u>PWF<math>\overline{\text{FAIL}}</math></u> ).
SR	Syndrome Register: A 7-bit register in the ATC that contains the Syndrome Bits arriving from memory via the <u>PMCHK</u> lines. The CPC can perform TOE and TIE operations on the SR.
Stack Pointer	A 5-bit incrementing/decrementing pointer in the CPC used to access the 32-entry Oper- and Stack portion of the Scratch Pad.
STAT Register	Status Register: / A 32-bit register in the SIC that contains the SIC transmission status.
State Register	/ A 16-bit register in the CPC which holds the information required to retry a Virtual Memory operation that resulted in a DAT No Match Interrupt.

TERM	DEFINITION/DESCRIPTION
SUR1-5	Set Up Register #1 to #5: Used by the CPC for special hardware assistance in decoding the Virtual Commands during specific virtual machine emulation.
Syndrome Bit	There are seven Syndrome Bits generated by the ATC and stored with each 32-bit word as check bits during Memory Store operations. They are loaded into the ATC SR during Memory Fetch operations and used for ECC verification.
System Clock	The circuitry in a NCR/32 VLSI Processor Family system which generates the 6.6 MHz, two phase (X0 & X1), System Clock from the 13.3 MHz input frequency. Each System Clock cycle has a period time of 150 nano-seconds. (See also CLOCK0)
Tally Register	A 16-bit decrementing register in the CPC used during Field operations to count the bytes being processed.
Test Mask Operands	Conditional Jump or Conditional Skip Instructions which test the Indicator Array or a byte in the RSU against the H, I, J, or K Field Mask. A transfer occurs when the test is valid.
TI	Trap Indicator: The indicator in Control Array #1 which disables the processing of traps.
TIE	Transfer In External: An external reference instruction executed by the CPC which may either read from an ERU or trigger a fetch from Scratch Pad memory.
TII	Transfer In Internal: A CPC internal transfer instruction which

TERM	DEFINITION/DESCRIPTION
	moves data from the Internal Register Unit (IRU) into an RSU.
TIP	<p>Transfer In Port:</p> <p>A CPC operation which transfers data from an ERU port (ERU64-127) into an RSU.</p>
TOD	<p>Time-Of-Day Register/Counter:</p> <p>An incrementing register in the ATC which can be used to store the current time and to generate timed interval interrupts.</p>
TOE	<p>Transfer Out External:</p> <p>An external reference instruction executed by the CPC to either write data to an ERU or to trigger a store to Scratch Pad Memory.</p>
TOI	<p>Transfer Out Internal:</p> <p>A CPC internal transfer instruction which moves data from an RSU to an IRU.</p>
TOP	<p>Transfer Out Port:</p> <p>A CPC operation which transfers data to an ERU port (ERU64-127) from an RSU.</p>
TRAP	<p>Trap:</p> <p>An input signal to the CPC indicating the presence of a trap condition.</p>
unpacked BCD	<p>Data representation in which a Binary Coded Decimal digit (least significant nibble) is combined with its ASCII digit character (most significant nibble) in a byte.</p>
VAR	<p>Virtual Address Register:</p> <p>A 32-bit register in the ATC that is loaded with the virtual address during Virtual Memory operations.</p>
VARB	<p>Virtual Address Register Buffer:</p> <p>A 32-bit register in the ATC that contains</p>

TERM	DEFINITION/DESCRIPTION
	the previous virtual address used for an address translation. It may be retrieved by the CPC if a DAT No Match Interrupt was set.
Virtual Address	Address asserted by the CPC which must be translated into a real address by the ATC. The translated real address is composed of the ATC Virtual Page Number and Page Displacement.
Virtual Indicators	A 16-bit register in the CPC which holds the flags and indicators relevant to Virtual Command execution.
Virtual Memory Reference	A memory access operation that requires the address be translated in the ATC. It may also be referred to as relative addressing. (See also Page Descriptor and Virtual Address.)
VLSI	Very Large Scale Integration: The term describing high density device layout using MOS cells.
VPN	Virtual Page Number: The 22-bit content portion of each of the 16 AM entries used in the DAT unit of the ATC.
VRX	Virtual Resource Executive: The hardware independent high-level Instruction Set for the NCR 8500 series systems.
Write Tag Register	A 4-bit register which controls the Byte Write Enables (PMWT0-3) during Virtual Memory Store operations.
Word Or Full-Word	32-bits of data in parallel or series.
LEGEND:	“n” indicates an arbitrary number.

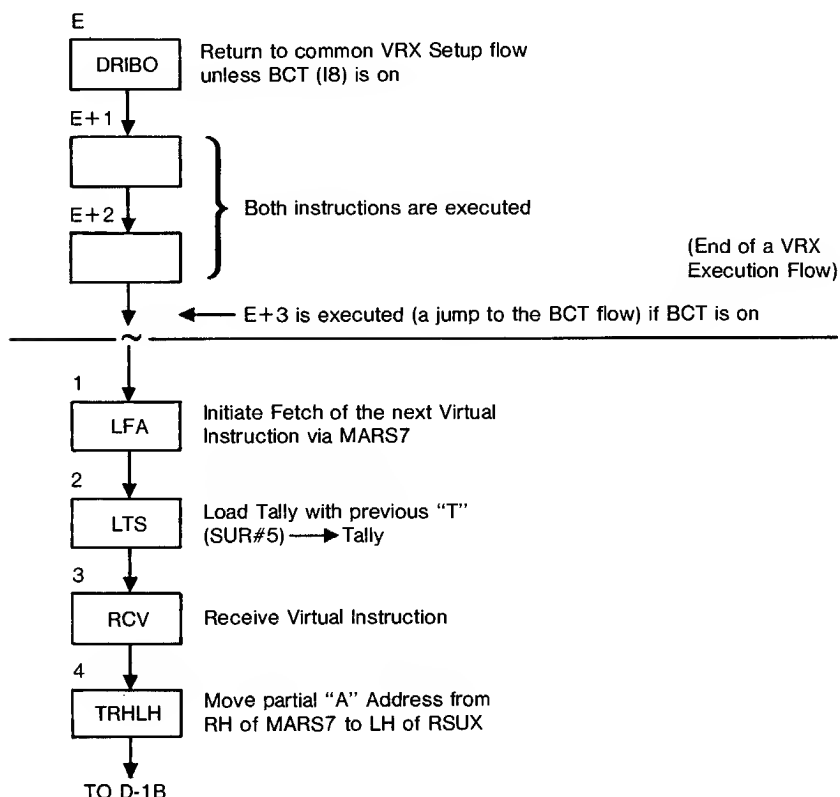
## APPENDIX B SETUP FLOWS

### SETUP FLOWS

The special Setup microinstructions in the Processor have been designed so that, for optimum performance, they must be used in precise sequences with other microinstructions. The flow diagrams in this appendix specify those sequences for the three explicitly supported Virtual Machines, and indicate which portions of the sequences are fixed and which portions are variable.

### VRX SETUP FLOWS

The VRX Setup Flows are depicted in Figures D-1A through D-3B.



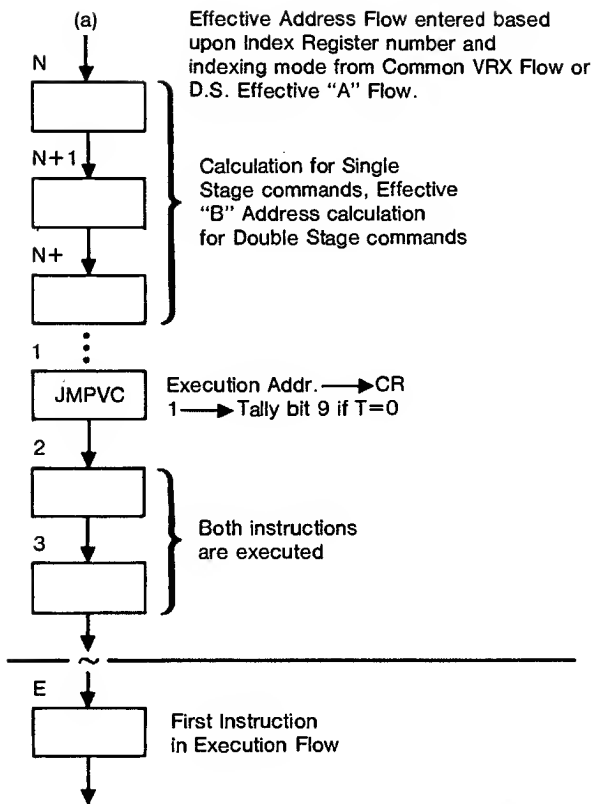
GIMB003

Figure B-1A VRX Common Setup Flow

**B-2**



**Figure B-1B VRX Common Setup Flow**



GIMB005

Figure B-2 VRX S.S. Effective "A" Flow and D.S. Effective "B" Flow

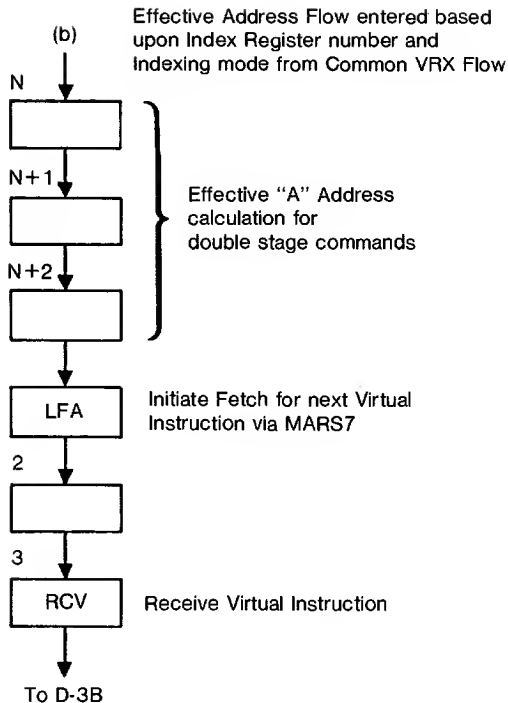


Figure B-3A VRX D.S. Effective "A" Flow

GIMB006

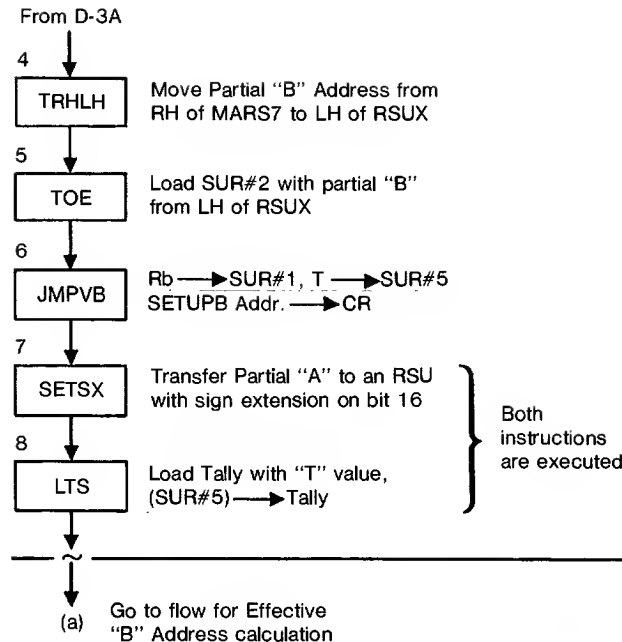


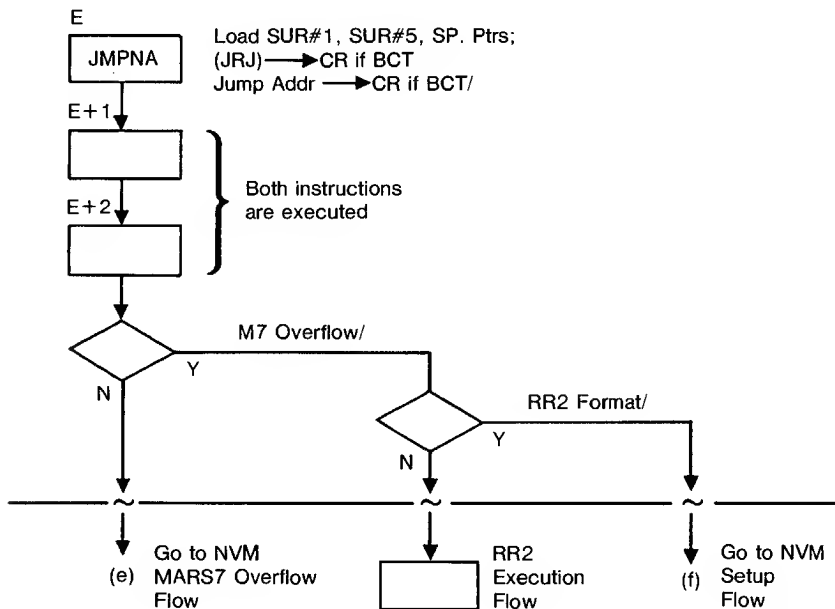
Figure B-3B VRX D.S. Effective "A" Flow

GIMB007



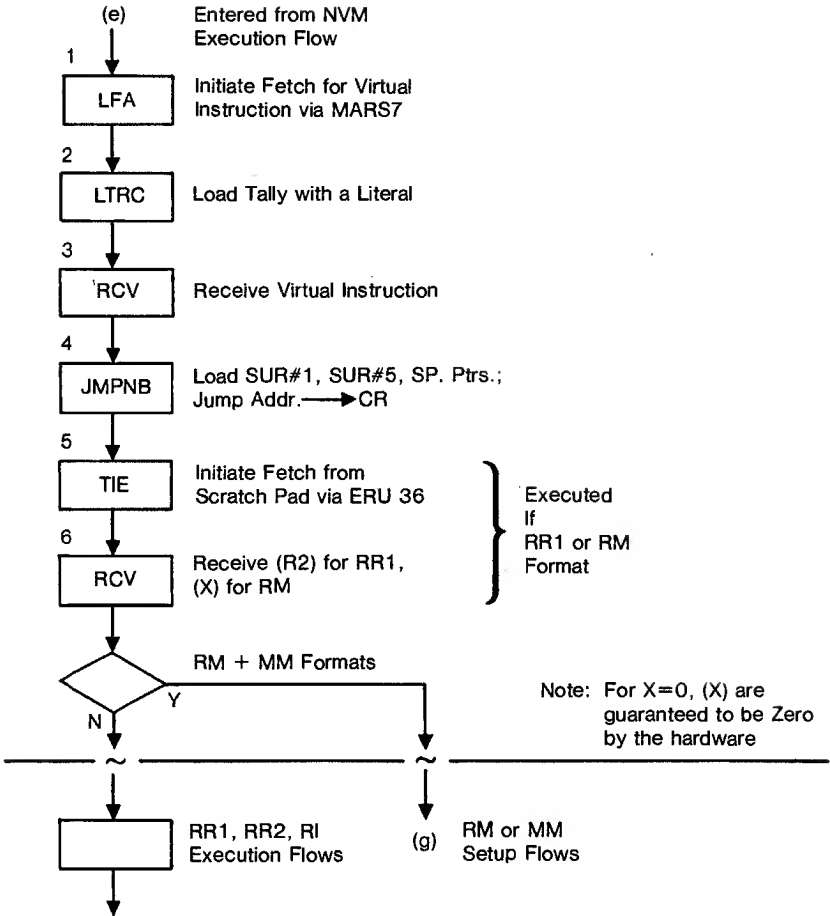
# NVM Setup Flows

The NVM Setup Flows are depicted in Figures D-4 through D-7C. The NVM Descriptor Setup Flow is shown in Figure D-8.



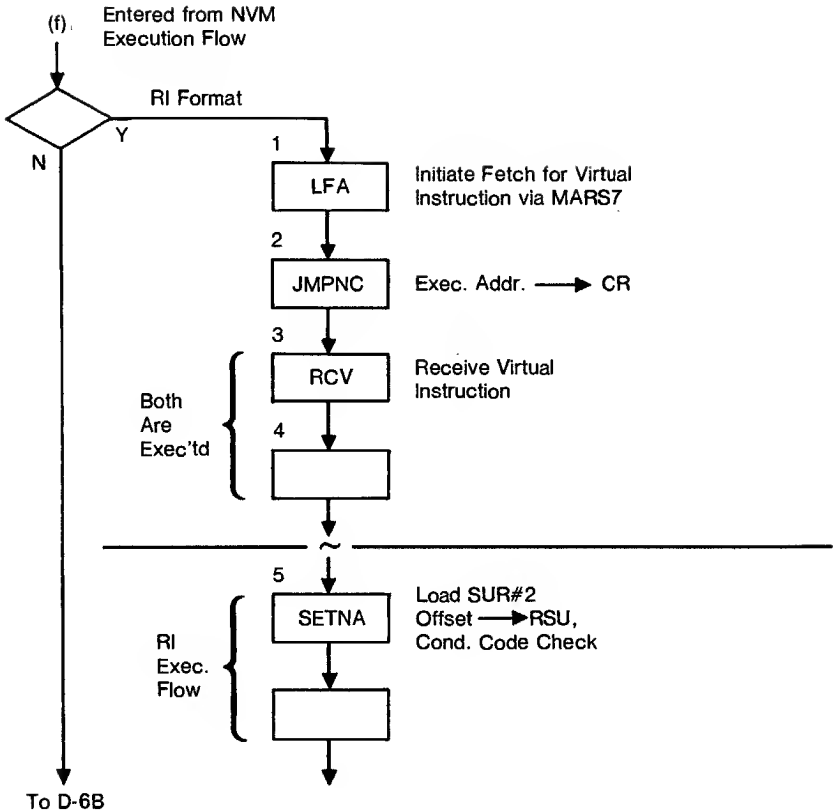
GIMB007-1

Figure B-4 NVM Jump from Execution



GIMB008

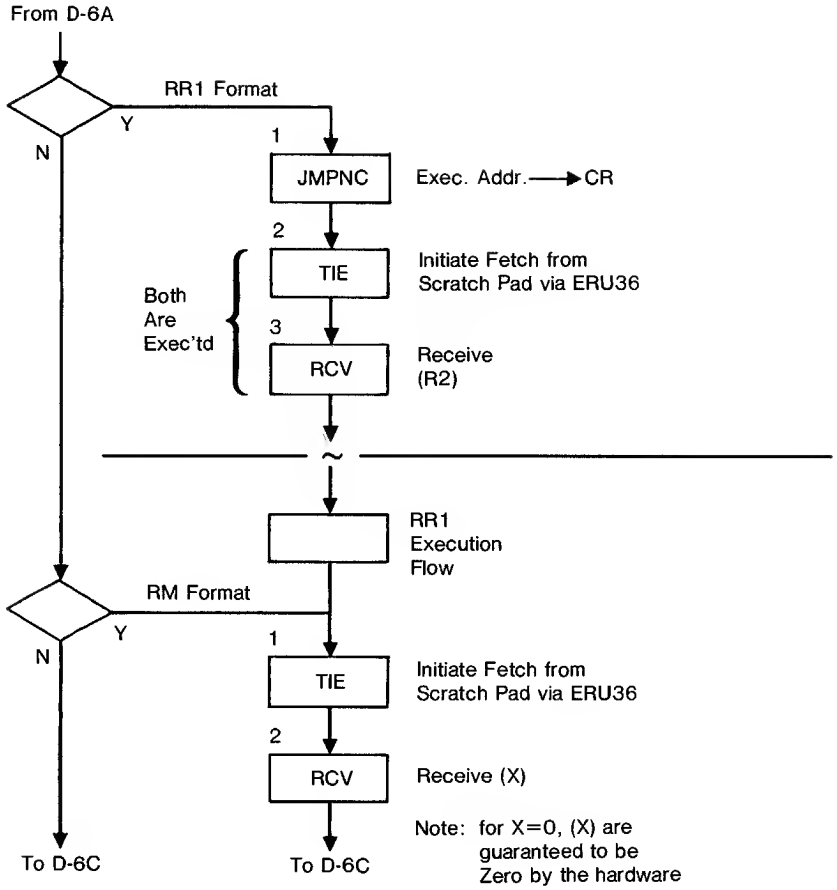
Figure B-5 NVM MARS7 Overflow Flow



GIMB009

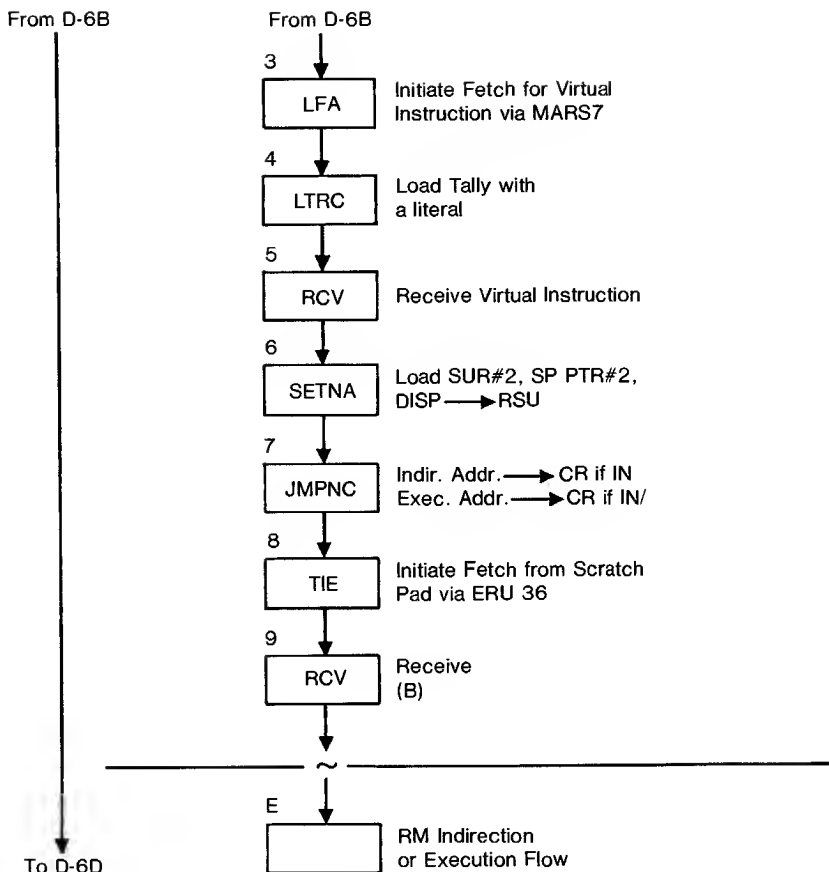
Figure B-6A NVM Setup Flows (from Exec.)

SETUP FLOWS



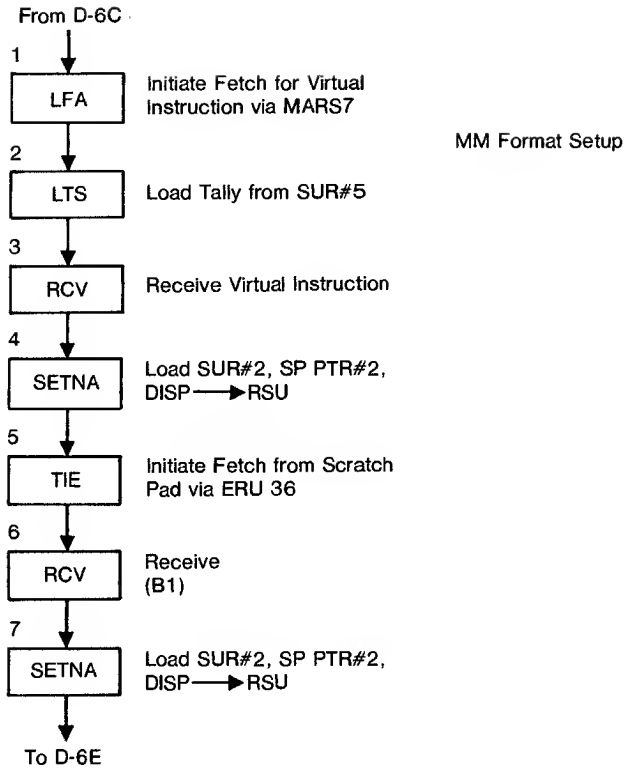
GIMB009-1

Figure B-6B NVM Setup Flows (from Exec.)



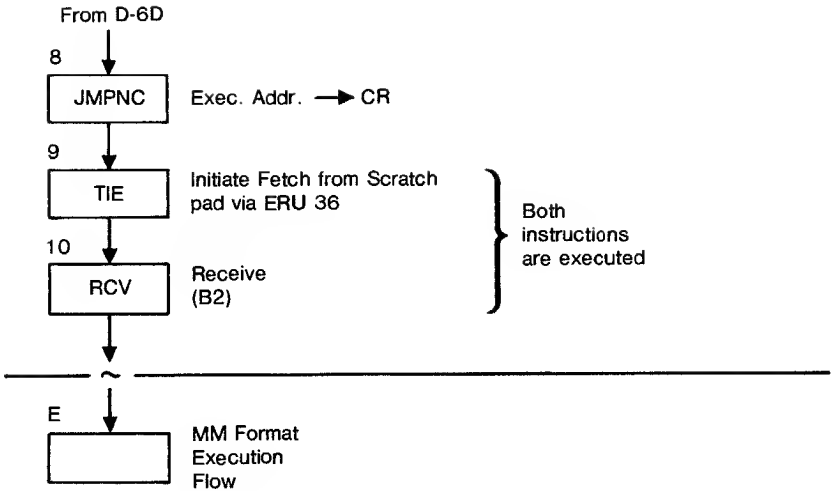
GIMB010

Figure B-6C NVM Setup Flows (from Exec.)



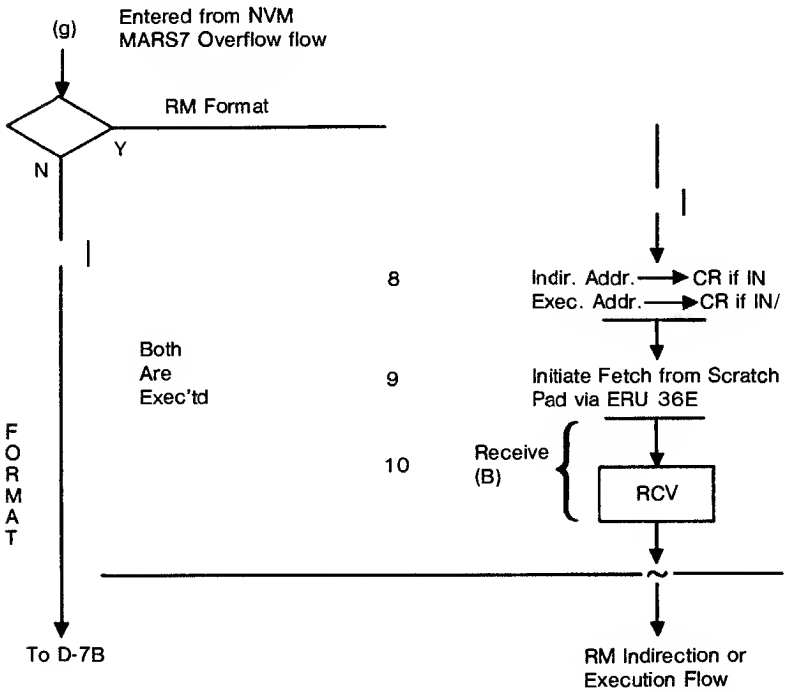
GIMB011

Figure B-6D      NVM Setup Flows (from Exec.)



GIMB012

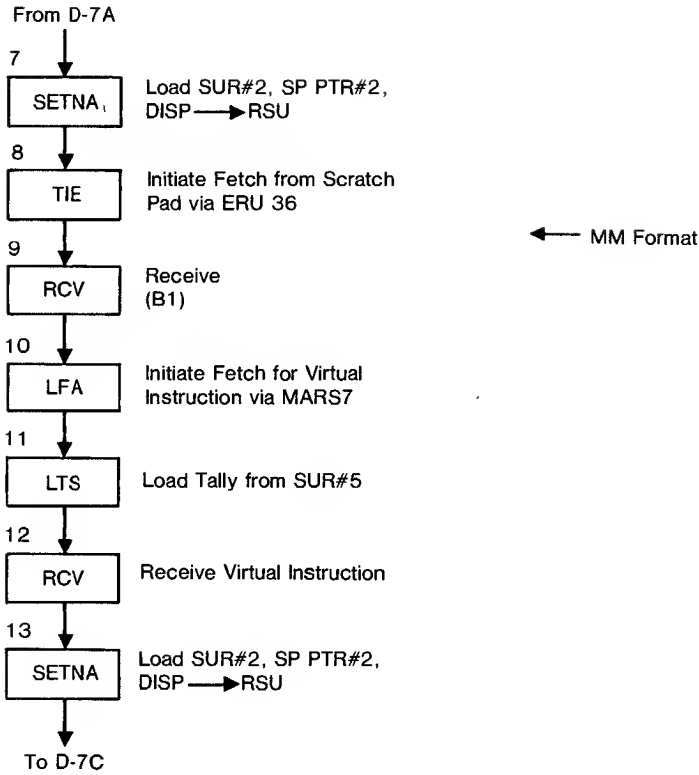
Figure B-6E NVM Setup Flows (from Exec.)



GIMB013

Figure B-7A NVM RM and MM Setup Flows (from Overflow)

SETUP FLOWS



GIMB014

Figure B-7B NVM RM and MM Setup Flows (from Overflow)



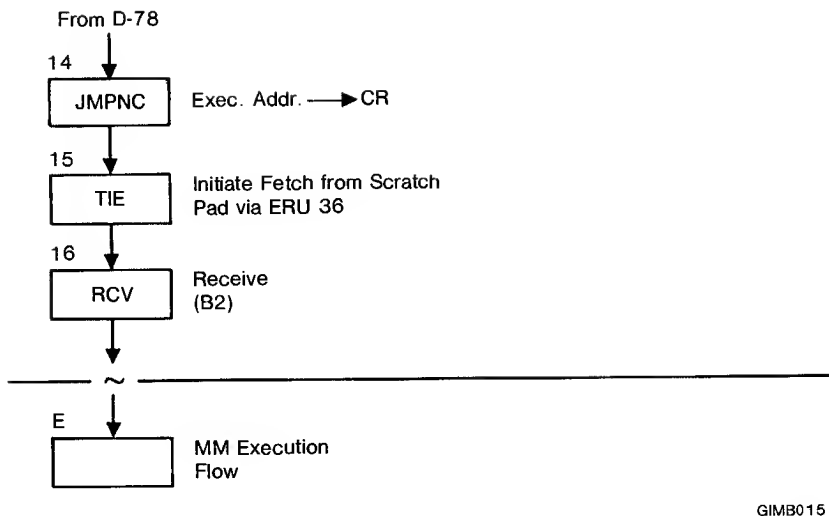


Figure B-7C    NVM RM and MM Setup Flows (from Overflow)

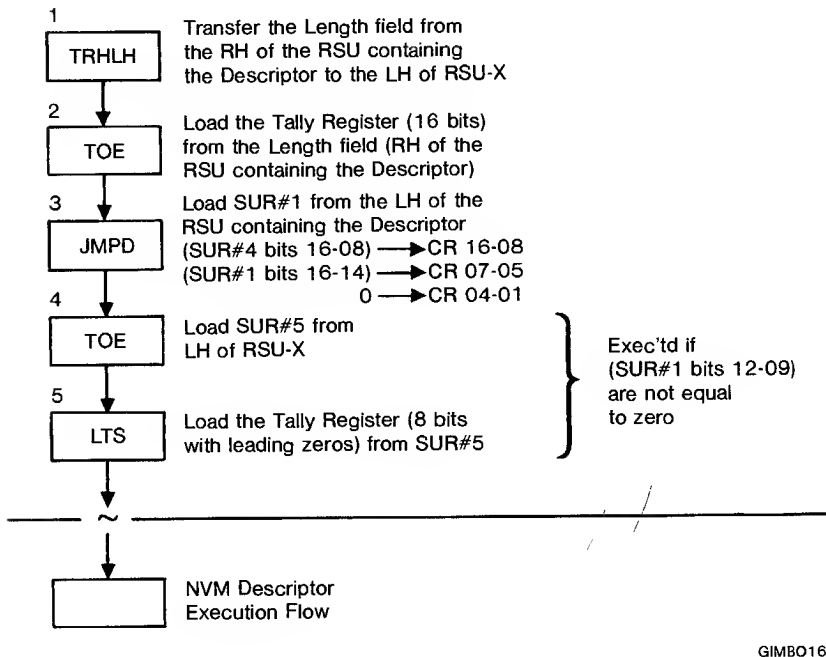
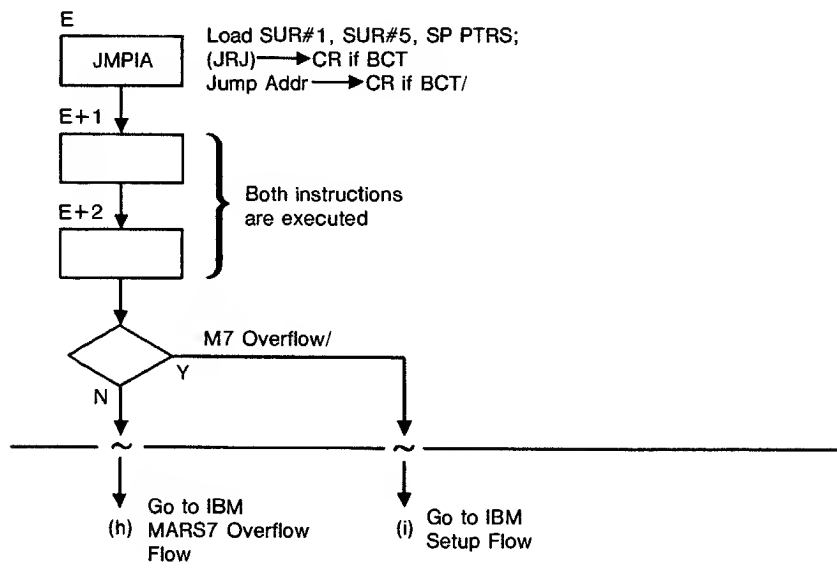


Figure B-8    NVM Descriptor Setup Flow

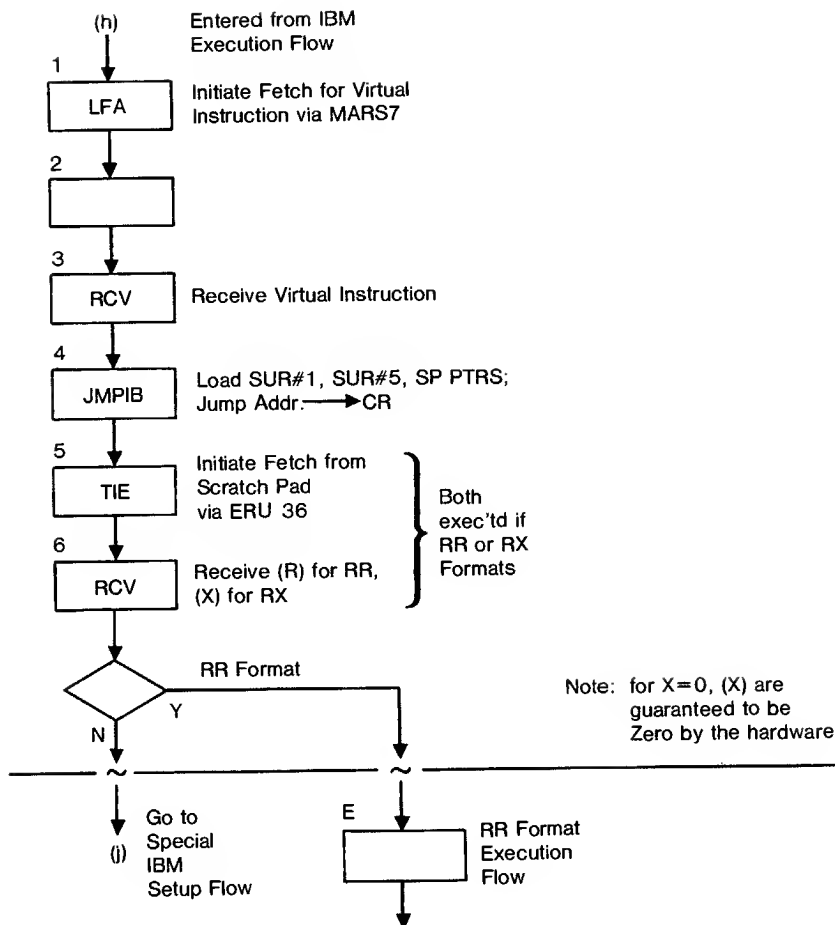
IBM SETUP FLOWS

The IBM Setup Flows are depicted in Figures D-9 through D-12C.



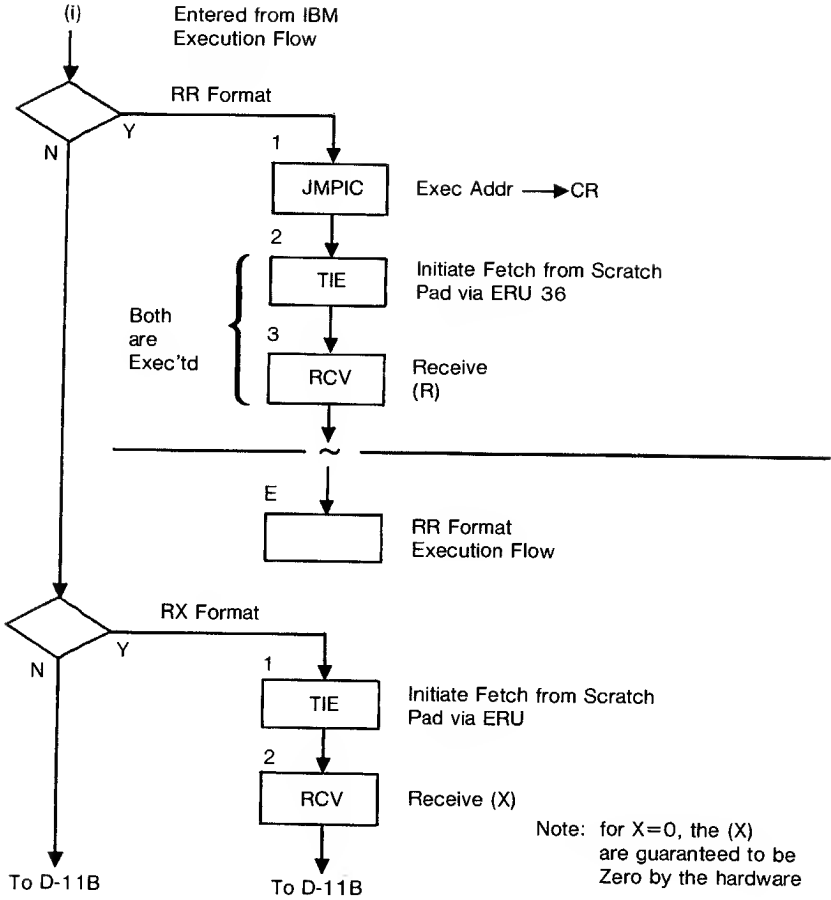
GIMB017

Figure B-9 IBM Jump from Execution



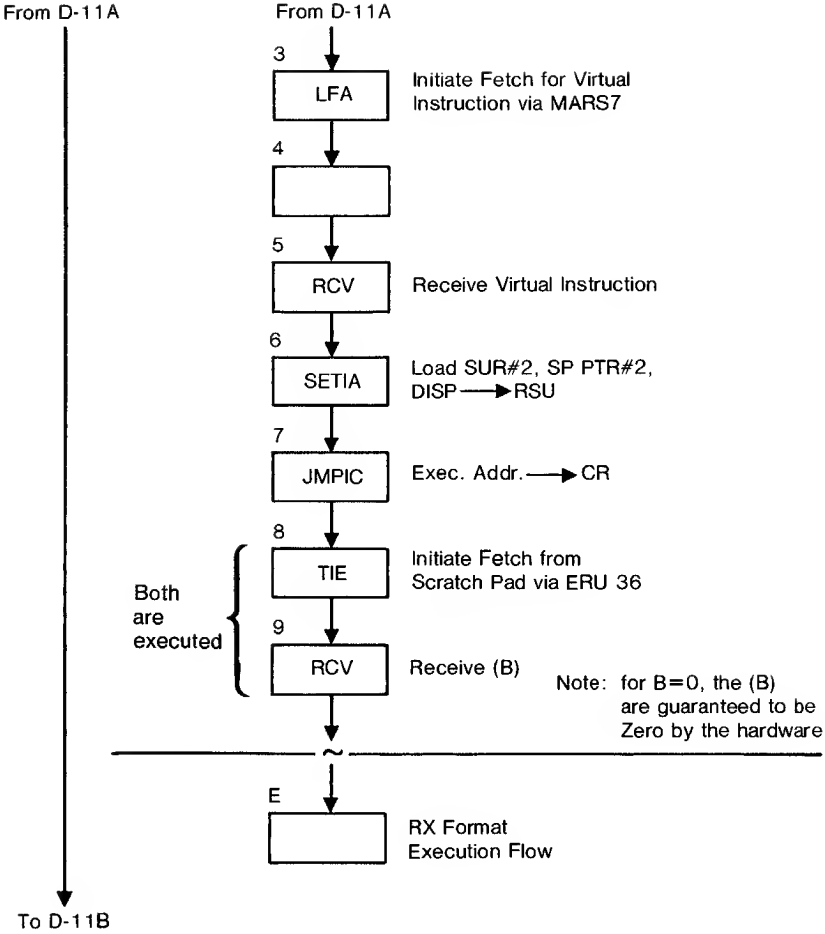
GIMB018

Figure B-10 IBM MARS7 Overflow Flow



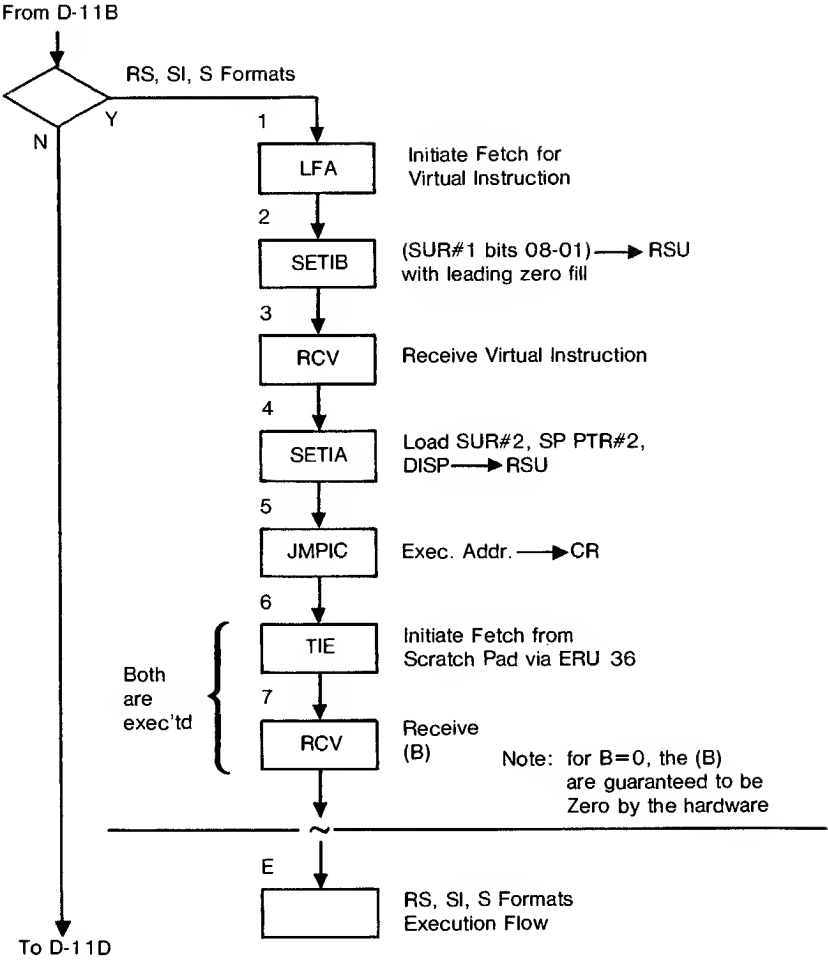
GIMB019

Figure B-11A IBM Setup Flows (from Exec.)



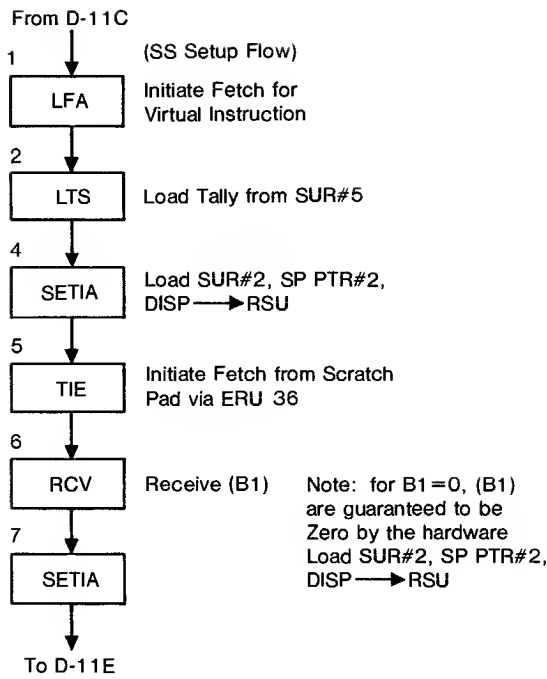
GIMB020

Figure B-11B IBM Setup Flows (from Exec.)



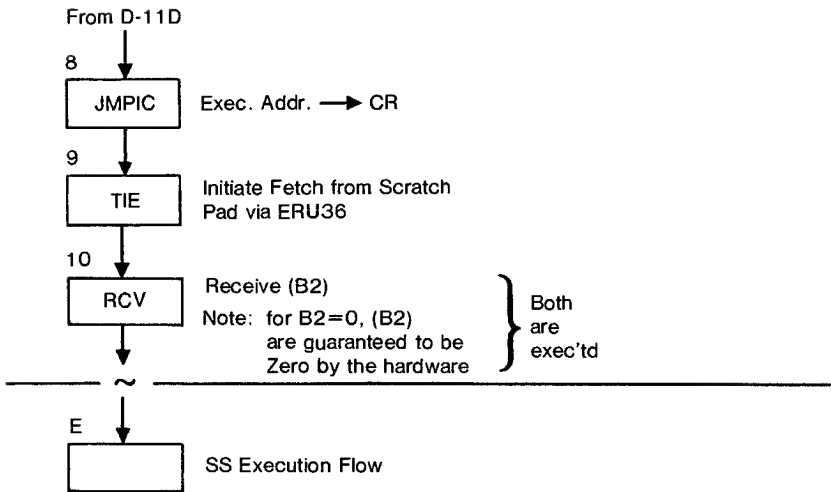
GIMB021

Figure B-11C IBM Setup Flows (from Exec.)



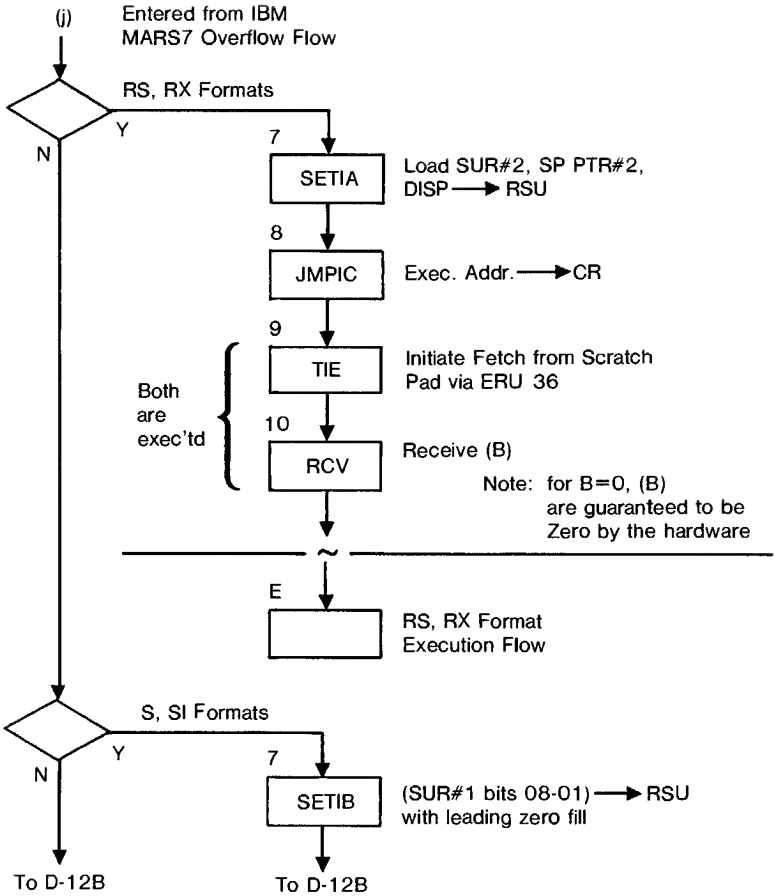
GIMB022

Figure B-11D IBM Setup Flows (from Exec.)



GIMB023

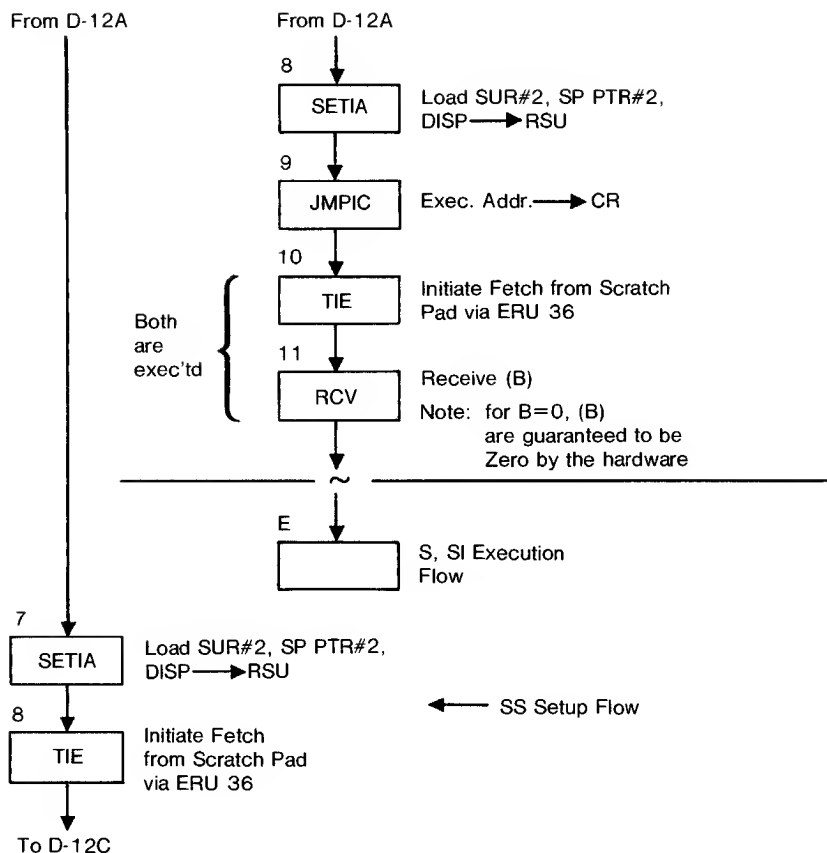
Figure B-11E IBM Setup Flows (from Exec.)



GIMB024

Figure B-12A IBM Setup Flows (from Overflow)

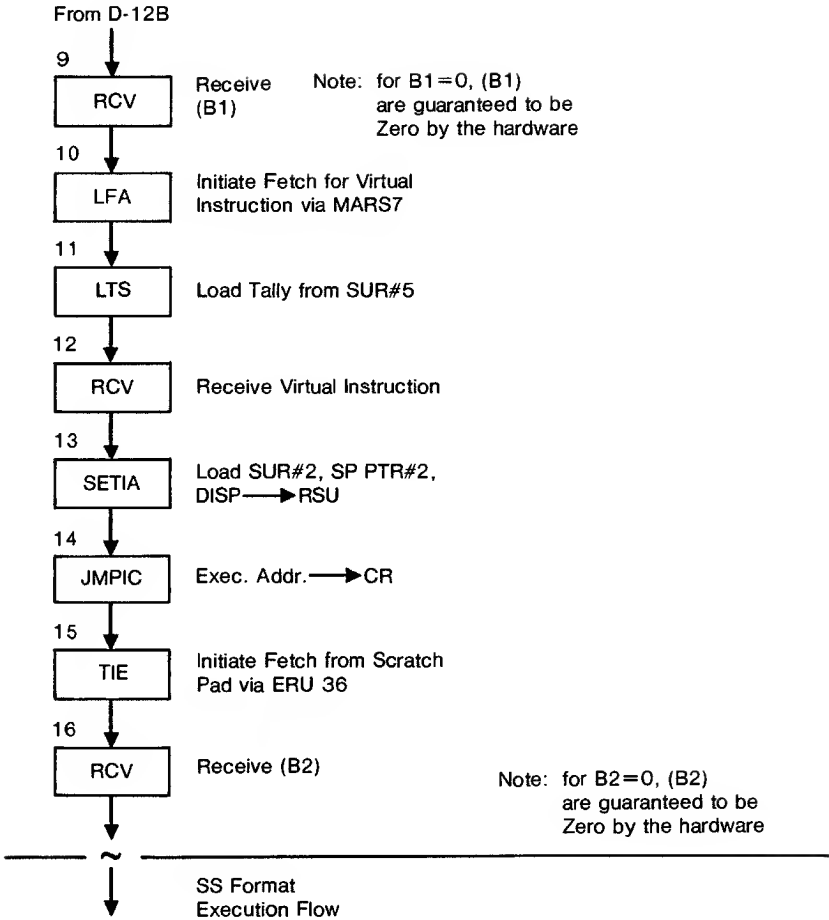




GIMB025

Figure B-12B IBM Setup Flows (from Overflow)

SETUP FLOWS



GIMB026

Figure B-12C IBM Setup Flows (from Overflow)

## APPENDIX C BREAKPOINT OPERATION

### BREAKPOINT OPERATION

This section describes an example of how to initialize, detect, and process breakpoints. Note, however, that external hardware must be designed and implemented for servicing breakpoints.

When the CPC is powered up or a System Reset is issued, all potential Breakpoints are disabled. A Breakpoint Enable bit is maintained in the Maintenance Control Register which can be read via the Maintenance Status Register. At the time of system initialization, hardware resets this bit to disable any spurious Breakpoints which might be present.

Each time that the Breakpoint routine is entered and Breakpoints are to be used, the Breakpoint Enable bit should be tested by a Breakpoint microinstruction routine. If the bit is clear, then the Breakpoint routine should proceed to clear all physical ISU locations of any potential Breakpoints. After purging the Breakpoint RAM, a TOP should be executed to the Maintenance Control Register to turn the Breakpoint Enable bit on.

A Breakpoint is cleared via the modified DJOR instruction. The ISU address at which the Breakpoint is to be cleared is placed into the right half of an RSU, then that RSU is addressed by the K-field of the DJOR instruction. J-field bit 01 is set to a one and J02 is set to a zero in the DJOR. The instruction following the DJOR must then be an unconditional immediate jump to void the jump which would occur from the DJOR unless, coincidentally, that jump is desired.

Likewise, a Breakpoint is set via the DJOR. Instead of setting J02 to a zero, it is set to a one.

### SAVING THE INTERRUPT FIFO

Both the interrupt and the trap service routines should be written to include a test of the Breakpoint Enable bit. If Breakpoints are not enabled, then the routine executes as it normally would. Two cycles (TIP and SRBZ) are required for the test.

If the Breakpoint Enable bit is on, then the contents of the FIFO should be saved and the FIFO restarted. In a trap service routine the Trap Indicator must also be reset. This allows both service routines to be trapped by a Breakpoint trap and provide a link back to the service routine.

### DETECTING BREAKPOINTS

External hardware on the Processor board detects the presence of a Breakpoint during the Fetch stage of a primitive instruction and tracks the Breakpoint, depending upon the specific hardware implementation, to either the Interpret Stage (X0) or to the Execute Stage (X0). If Breakpoints are enabled, the hardware sets the Breakpoint trap bit in the Maintenance Status Register and sources the TRAP/signal to the CPC.

When the trap is detected, the Firmware routine transfers in the Maintenance Status Register and tests the Breakpoint trap bit. If the bit is off and the Breakpoint Enable bit is on, then the FIFO is saved as described previously. If the Breakpoint trap is on, then the FIFO is not saved even though the Breakpoint Enable bit will be on. Since the Breakpoint trap executes similar to other traps, the FIFO will have been stopped, and TI turned on. The trap is cleared by a transfer out to the Maintenance Control Register with the Breakpoint trap bit off.

If the hardware implementation tracks the Breakpoint address to the Execute Stage, then the instruction at that address will be executed provided a previous instruction did not establish a skip condition. The standard RTI sequence is used to Restore from the Breakpoint trap.

If the hardware implementation tracks the Breakpoint address to the Interpret Stage, then the instruction at that address will not be executed until the Restore from the Breakpoint routine is performed. That instruction will be the first instruction executed following the three RTI instructions.

The Breakpoint routine can (in the latter implementation only) determine whether or not that the Breakpoint instruction will be executed upon Restoring by testing the Skip Count in Control Array #1. Any non-zero count indicates that the Breakpoint instruction will subsequently be aborted.

### RETURN FROM BREAKPOINT ROUTINE

The return from the Breakpoint routine is performed via the FIFO using three RTI instructions. When returning to an interruptable routine the sequence is the same as for any trap. When returning to a non-interruptable routine, as determined by a test of NIE, an RC instruction is executed to reset TI and then the three RTI instructions are executed. The second RTI will not have control bit 01 set, though, as normally is the case since NIE should not be set on.

The first RTI of the Restore sequence for an Interpret Stage tracked Breakpoint must have bit J05 of the instruction set on. This is the Breakpoint enable control for setting or resetting Break-

points. Since the RTI instruction will present the Breakpoint address to the detection logic again, the J05 enable control being on prevents the same Breakpoint from occurring upon the Restore from itself.

Bit J06 (=1) can be used to reinforce the Breakpoint bit if it is intended to leave a Breakpoint at that address or it can be used (J06=0) to clear the Breakpoint at that address.

## PLACEMENT OF BREAKPOINTS

Using the trap method for Breakpoints implies that the return from a Breakpoint will be to the next logical instruction following the Breakpoint. The strategy, then, in placing Breakpoints in a program is to locate a Breakpoint at the last instruction that is desired to be executed. Since the return is not to the Breakpoint instruction, but rather to the next instruction, there is no problem with leaving a Breakpoint set after taking that Breakpoint. As indicated in F1.3, it is possible to test the skip count to determine whether or not the next instruction will be executed upon the return from Breakpoint and then if desired, to abort the Breakpoint routine.

## RESTRICTIONS ON THE USE OF BREAKPOINTS

Breakpoints cannot be set during any routines where the FIFO is not active if recovery is intended. This includes the Breakpoint service routine and the front end of the interrupt and trap service routines prior to the saving of the FIFO.

In debugging trap routines (other than the Breakpoint routine) using the Breakpoint scheme where TI is turned off, it will become necessary to disable interrupts via the Interrupt Mask Register if NIE is on.

The remainder of the Breakpoint restrictions are system dependent based upon whether the Breakpoint is tracked to the Interpret Stage (instruction not executed) or to the Execute Stage (instruction executed).

## APPENDIX D

### MEMORY RETRIES

#### MEMORY RETRIES AFTER NO-MATCH INTERRUPTS

A Virtual Memory Store or Fetch operation is aborted and a No-Match Interrupt generated if an Address Translation Fault occurs. Since the Virtual Store and Virtual Fetch instruction sequences are inherently different, and since distinctions between Stores and Fetches should ideally be transparent to the No-Match Service Routine, some hardware facilities have been added to the CPC to facilitate DAT fault recovery.

The Virtual Store sequence involves executing a Store, Store and Augment, or a Store and Decrement instruction (with AT on) followed by any interruptable instruction. The Virtual Fetch sequence involves executing a Fetch, Fetch and Augment, Fetch and Augment with Linkage, or Fetch and Decrement instruction (with AT on) followed by an interruptable instruction followed by a Receive Fetched Data (RCV) instruction. A No-Match interrupt on a Virtual Store operation will occur during the instruction after the Virtual Store instruction. On a Virtual Fetch operation the interrupt will occur during the Receive Fetch Data instruction. In both cases the instruction in the execution stage at the time of the interrupt will be executed.

A retry of the aborted memory operation is performed during the interrupt service routine by executing a program sequence that is identical for Stores or Fetches. Special retry hardware is used to make the required distinctions. That sequence consists of a Memory Reference Retry (MRR) instruction followed by an interruptable instruction followed by a Received Fetched Data instruction (with bit 2 of the K-field set to a one).

At the time the original Virtual Store or Fetch instruction executes, the Protection Code information, the Source/Destination Data RSU Address and the Write Tags are all saved in the State Register. The State Register is not clocked again until the next Virtual Memory instruction or RCV instruction is executed. The RCV instruction clocks its Destination RSU Address into the State Register, though, only if bit 01 in the K-field is on.

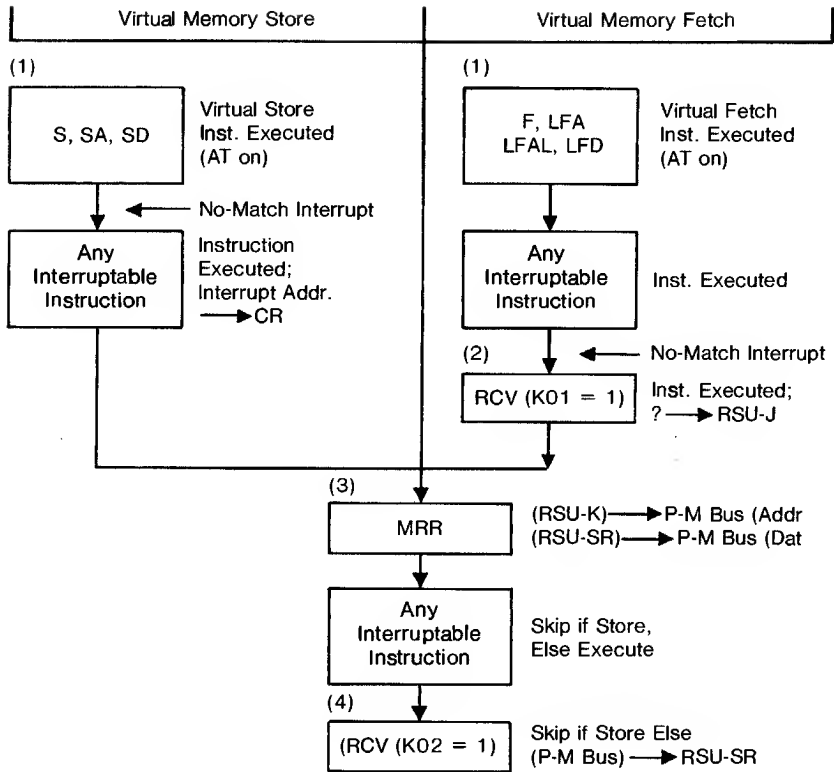
The MRR instruction retriggers the memory reference portion of the Store or Fetch that caused the No-Match interrupt. The address augment or decrement has already been accomplished. MRR uses the contents of the RSU specified by the K-field as the Virtual Address for the memory operation. The Data RSU Address saved in the State Register is used to specify a Source Data RSU (note that

this data is relevant only on Store operations and that the instruction which follows any store instruction which might generate a virtual interrupt cannot alter this data). The Write Tags are supplied from the State Register as well as the Protection Code which acts as the eventual Store or Fetch designator.

During execution of the MRR instruction the hardware decodes the Protection Code to determine if a Store operation is being performed. If so, skip controls are set which cause the subsequent two instructions to be voided. For a Fetch operation these two instructions are executed. The RCV instruction in this sequence must have bit 1 of the K-field set off and bit 2 of the K-field set on. This will prevent the State Register from being altered while at the same time allowing the RSU Address saved in the State Register to specify the Destination RSU for the fetched data (overriding the J-field).

The resulting Virtual Memory operation may result in another No-Match Interrupt if Firmware has not successfully created the Associative Memory entry, or an Access Violation Interrupt if a protection check error was detected. Either interrupt will not be recognized by the Processor until the Restore from Interrupt Sequence has been completed at which time the Normal Interrupt Enable control is turned back on. The program will be restored to execute first the same instruction which was in interpretation at the point of the original interrupt. If a DAT interrupt is then pending, the instruction will not be executed until after the completion of the subsequent interrupt routine. The same information required to perform the retry on the first interrupt will still be present in the State Register for the second interrupt.

Refer to Figure B-1 for a flow chart of the Virtual Memory Retry sequence.



NOTES:

1. Protection code, RSU-K0 Address and Write Tags saved in State Register
2. RSU-J Address saved in State Register if K01 = 1
3. Protection code, RSU Address, Write Tags supplied from State Register
4. RSU Address supplied from State Register if K02 = 1

GIMB001

Figure D-1 Virtual Memory Retry Sequence



## APPENDIX E

### FETCHING FROM ISU

#### FETCHING FROM ISU

The CPC instruction set does not support an explicit instruction which allows fetching of data (tables, etc.) from the ISU. However, a sequence exists which effectively performs the same function. This sequence depends upon the use of a two word instruction beginning as the second instruction located beneath an unconditional delayed jump. Ordinarily this is considered a violation of a restriction involving delayed jumps since the literal used by the two word instruction will not follow the coded flow. In realizing what the hardware does in this circumstance, though, the restriction can be overlooked and used to effect a Fetch from Control Store.

The sequence to produce a Fetch from Control Store is:

1. Delayed Jump
2. Delayed Jump
3. LRH or LRHC
4. X

The Delayed Jump in line 1 should be able to specify the desired Control Store Address to be fetched from (e.g., DJOR, DRIBO, etc.). The Delayed Jump in line 2 should return the program flow to the desired instruction code (could be instruction 4). The LRH or LRHC instruction when it executes will pick up the instruction, to be used as a literal, that is in the pipeline interpret stage while the LRH or LRHC is in the execute stage. With the above sequence, that instruction (literal) will be the instruction referenced by the delayed jump in line 1.

Thus, by varying the address specified by the first delayed jump, this sequence can be used to Fetch any number of words from ISU.

## APPENDIX F

### NON-INTERRUPTABLE INSTRUCTIONS

#### NON-INTERRUPTABLE INSTRUCTIONS

The following instructions are non-interruptable.

#### OP CODE

(HEX)

#### INSTRUCTION

02	MEMORY REFERENCE RETRY
03	FETCH (REAL)
04	FETCH
05	LOAD, FETCH AND AUGMENT
06	LOAD, FETCH AND AUG. (W. LINKAGE)
07	LOAD, FETCH AND DECREMENT
08-0B	FETCH (LITERAL)
00,01	TRANSFER IN EXTERNAL (ERU32-63)

NOTE: Any instruction that references the PM Bus is non-interruptable during the period that the Bus is unavailable.

## APPENDIX G ARRAY MATRICES

### CONTROL ARRAY AND INTERRUPT/TRAP ARRAY MATRIX

Control Array and Interrupt/Trap Array Matrix for Virtual Memory Operations.

Control Array Bits

Int. Array Bits	CA1 (PRV)	CA2 (BPE)	CA3 (ME)	CA4 (VME)	CA5 (AS)	CA6 (NPC)	CA8 (STE)	Virt. (1) Oper.
ITA7 (MI)	X	X	1	0	(2)	X	0	Store (c)
ITA7 (MI)	X	X	0	X	X	X	1	Store (u)
ITA8 (VBPI)	X	1	X	0	(2)	X	X	M7 Fetch (c)
ITA9 (CPI)	X	X	X	X	(2)	X	X	Store (3), (4)
ITA10 (NMI)	X	X	X	X	(2)	X	X	Fetch/Store (5)
ITA11 (AVI)	(6)	X	X	X	(2)	0	X	Fetch/Store (4)
ITA6 (VMI)	X	X	1	1	(2)	X	X	Store (C)

Notes:

- (1) AT must be on for all virtual operations
- (2) Address comparisons are dependent upon this bit
- (3) Page is being written into for the first time
- (4) This interrupt is dependent upon a successful Associative Search
- (5) This interrupt occurs on an unsuccessful Associative Search
- (6) This set of Protection Check bits used is dependent upon this bit
- (c) = Conditional interrupt (dependent upon virtual address matching Address Monitor Register contents)
- (u) = Unconditional interrupt
- X = Don't care

GIMTEB002

## APPENDIX H

### PBCD AND UBCD SETTING

Conditions Setting Indicators PBCD and UBCD

CPC Instr.	Set I5 If:	Set I5 If:	Set I6 If:
	LD not 0-9	RD not 0-9	LD not 3
TLDL	X	X	X
TLDR	X	X	X
TRDL	X	X	X
TRDR	X	X	X
TFFD	X	X	X
TFFI	X	X	X
TBF	X	X	X
TBFD	X	X	X
TBFDN	X	X	X
TBFIN	X	X	X
TFBIN	X	X	X
TBFI	X	X	X
TFBD	X	X	X
TFBDN	X	X	X
TFBI	X	X	X
TFB	X	X	X
APDB	X	X	X
APDBC	X	X	X
APDF	X	X	X
AUDF		X	X
SPDB	X	X	X
SPDBC	X	X	X
SPDF	X	X	X
SUDF		X	X
CFU	X	X	X
CBFU	X	X	X

Note: PBCD and UPBCD are set by Hardware, and once set, must be reset by Firmware.

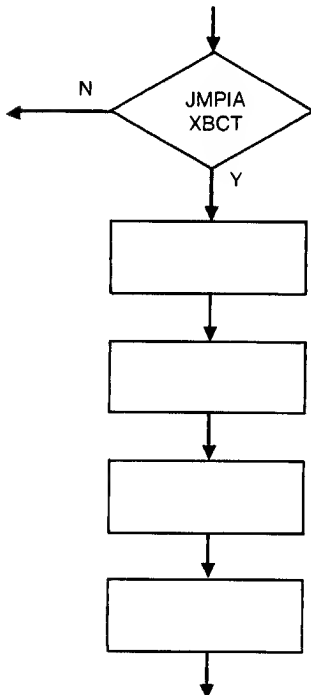
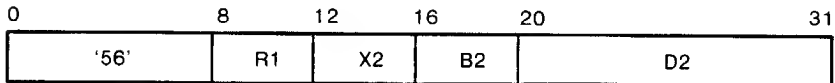
GIMTEB027

## APPENDIX I INSTRUCTION EMULATION EXAMPLE

This appendix describes the detailed flow for the execution of the IBM "OR" instruction in RX (register-and-indexed storage) format by the NCR 32 bit VLSI Chip Set. This IBM instruction requires a minimum of 3.6 microseconds and a maximum of 6.0 microseconds for execution, when the system cycle time is 150 ns per micro-instruction.

The detailed microinstruction flow of this IBM "OR" instruction is shown below.

Detailed flow for: O R1,D2(X2,B2) [RX]



Is Next Command already present? (If the last command ended on a halfword boundary, the first halfword of the next command is already present. If the last command ended on a fullword boundary, the first halfword of the next command is not present).

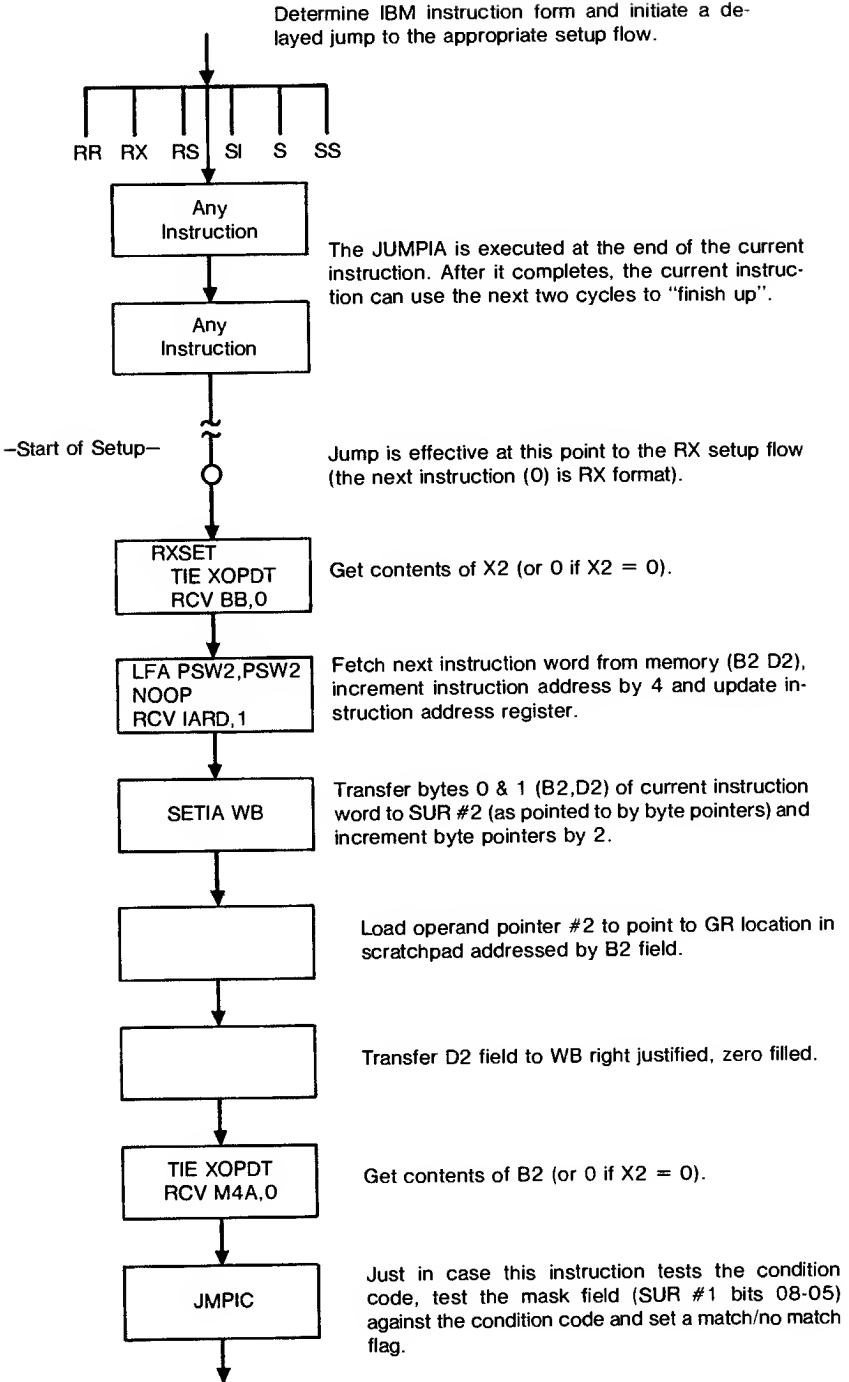
Transfer bytes 2 & 3 of current instruction word (first halfword of next command) to SUR #1 and increment byte pointers by 2.

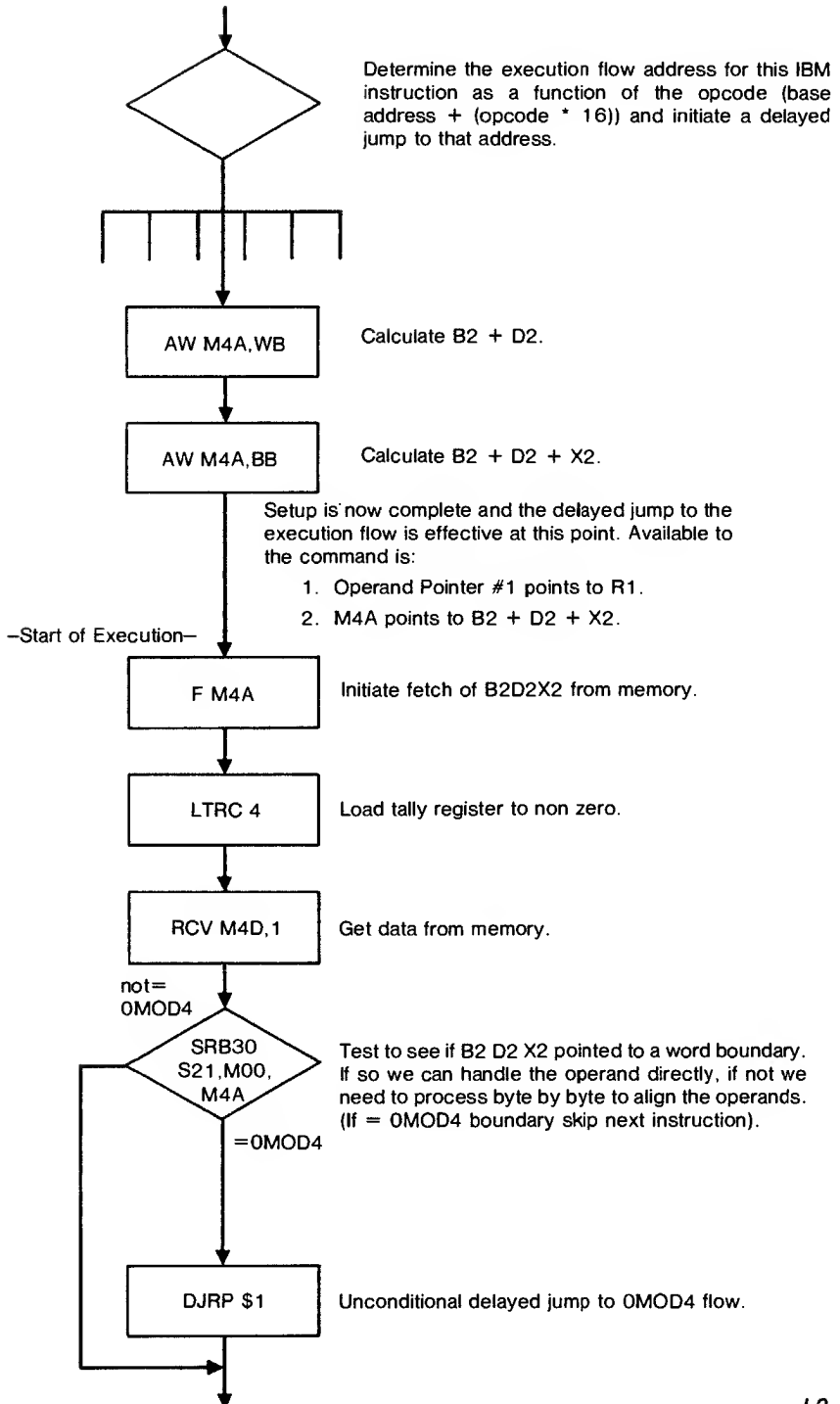
Transfer byte 3 of current instruction word to SUR #5 (to be used for tally if needed).

Load operand pointer #1 to point to GR location in scratchpad from R1 field.

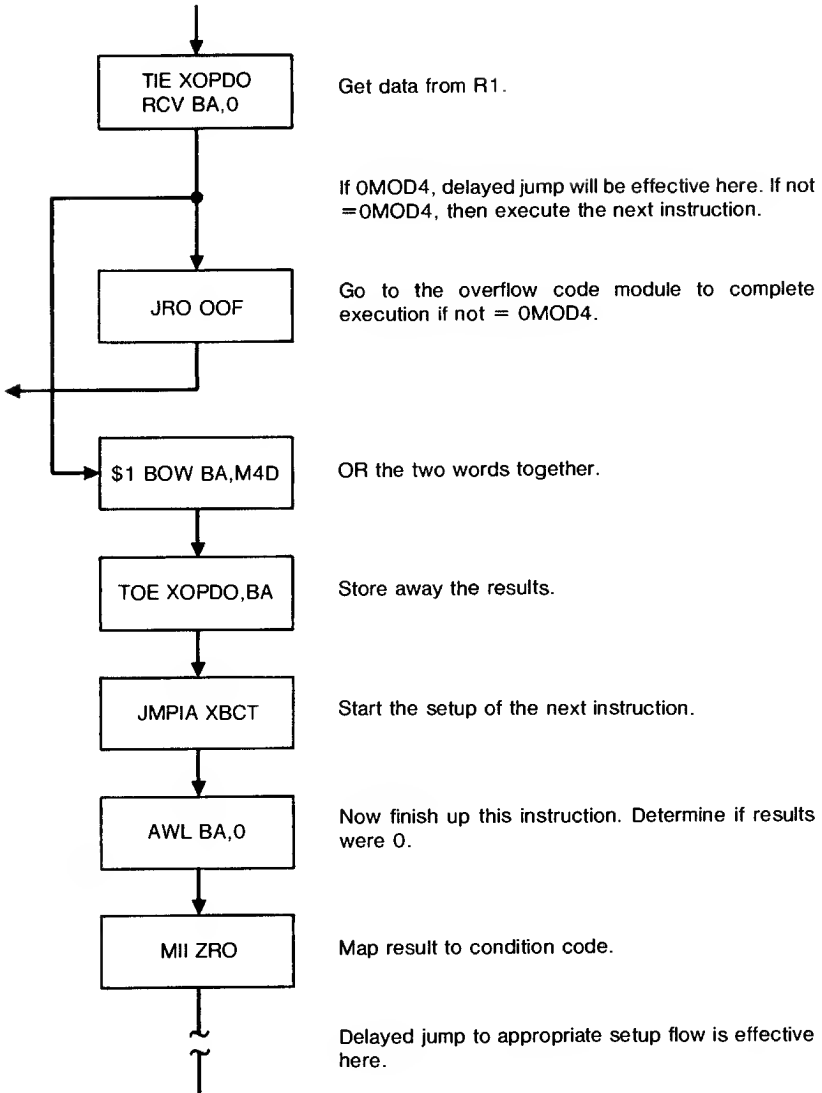
Load operand pointer #2 to point to GR location in scratchpad from X2 field.

# INSTRUCTION EMULATION EXAMPLE





## INSTRUCTION EMULATION EXAMPLE





**DEFINITION OF TERMS**

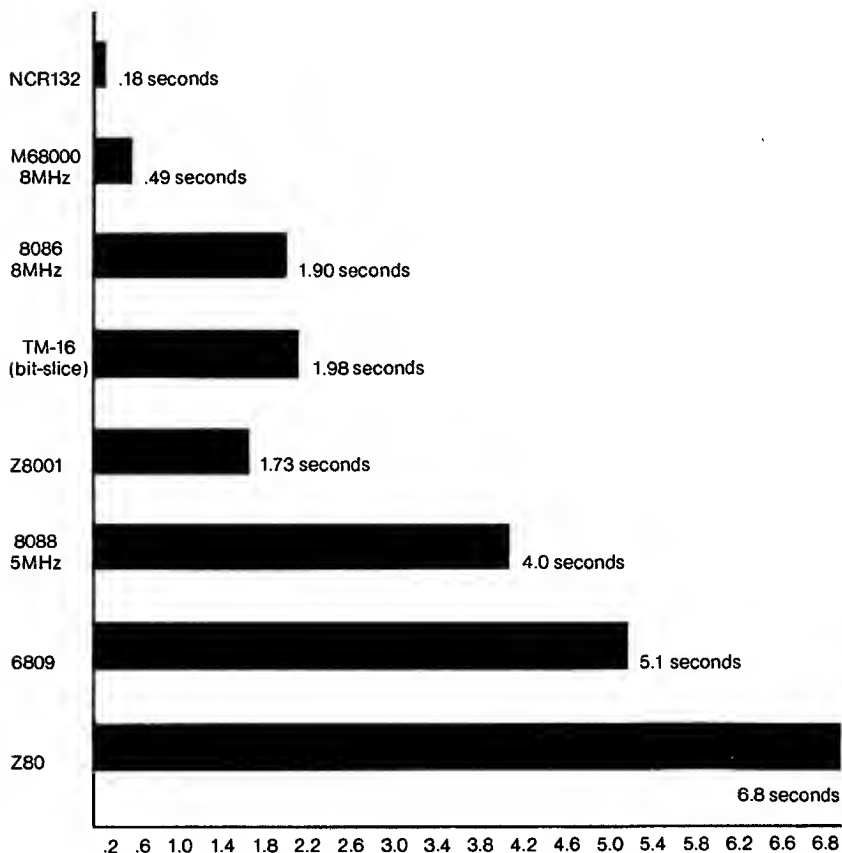
XBCT	—	The address of the routine to process between commands testing.
XOPDO	—	Equate for operand data #1 (ERU 32).
XOPDT	—	Equate for operand data #2 (ERU 35).
BA	—	Equate for byte addressable internal register 2.
BB	—	Equate for byte addressable internal register 3.
PSW2	—	Equate for MARS 7 address register (register 14).
IARD	—	Equate for MARS 7 data register (register 15).
WB	—	Equate for word addressable register 5.
M4A	—	Equate for MARS 4 address register (register 8).
M4D	—	Equate for MARS 4 data register (register 9).
S21	—	Equate to select bits 2 and 1.
M00	—	Equate to select both bits off.
ZRO	—	Equate to cause mapping of =0 to IBM condition codes.

## APPENDIX I

### SUMMARY

The Sieve of Eratosthenes has recently become a popular benchmark for microprocessor performance. The Sieve algorithm has been implemented on the NCR/32 chip set using two completely different approaches. The two approaches are used to illustrate various aspects of microprogramming the NCR/32. Execution characteristics and optimization techniques are also discussed.

The performance of the NCR/32 is also discussed. The execution time of the NCR/32 for ten iterations of the Sieve is compared with similar values published in BYTE magazine for other commercial microprocessors. As shown in the following graph, the NCR/32 outperforms the nearest competitor by a factor of 2.75.



## INTRODUCTION

The Sieve of Eratosthenes is a classical algorithm for finding prime numbers. This algorithm was developed in the third century B.C. by Eratosthenes, a Greek mathematician. More recently, the Sieve has become a popular benchmark for estimating performance of microprocessors, compilers, and high level languages.

The Sieve algorithm assumes the existence of an array representing the odd natural numbers. The even numbers are not included, since they are divisible by two. The algorithm eliminates the non-prime numbers by crossing out all the multiples of the prime numbers within the array. First, the multiples of three are eliminated by crossing out the third number following three, the third number following that number, and so forth, until the end of the array is reached. Next, all multiples of each remaining prime number are crossed out in the same manner, until only the primes remain.

The Sieve benchmark, as defined in the BYTE magazine articles of September 1981 and January 1983, specifies 10 iterations of the algorithm and an array size of 8190. Thus, the resulting program finds the number of prime numbers between 3 and 16361 ten times.

The Sieve is a valuable performance benchmark for several reasons. First, the algorithm involves no multiplication or division. Thus, machines without native multiply/divide commands are not handicapped. Second, the benchmark is memory intensive, giving a broader estimation of system performance. Third, the algorithm is very straightforward, thus simplifying the generated code and minimizing the impact of coding efficiency.

A slight variation of the Sieve algorithm notes the fact that any prime which is greater than the square root of the largest number in the array will have no multiples within the array. Program execution time can be reduced by comparing each prime number with this square root. If the prime is greater than the square root, the routine which eliminates the multiples of the prime is skipped. The implementation of this algorithm generally results in a performance improvement of approximately 25 percent.

## NCR/32-000 IMPLEMENTATION

The Sieve algorithm has been implemented in NCR/32-000 microcode using two different coding approaches. Both approaches use an array of memory words as the flag array. However, while the first and simplest approach uses each memory WORD as one flag, the second approach uses each memory BYTE as a flag. The second approach reduces the time spent on memory accesses, but requires more complex code utilizing the field commands of the NCR/32.

## WORD-FLAG APPROACH

The word-flag approach uses a memory array of 8190 entries. Each memory word represents an odd number between zero and 16,361. Non-prime numbers are identified by 32-bit flags. This approach is simpler and less efficient than the byte-flag approach.

### Program Flow

The program flow for the word-flag approach is illustrated in Figure 1. The code is divided into six functional blocks and five conditional branches. For ease of referral, the functional blocks will be assigned numbers, as follows:

Block 1: Initialize Registers.

Block 2: Decrement Iteration Count.

Block 3: Clear Flag Array.

Block 4: Load Word and Augment Pointer.

Block 5: Increment Prime Number Count.

Block 6: Cross Out Next Multiple.

Block 1 initializes the CPC registers for entry into the routine. It loads the dimensions and location of the memory array, the literal value of the flag, the number of iterations, and a jump address. None of these values need to be reset for successive iterations of the Sieve routine. Block 1 is listed in Figure 2. Note that the term "LIT" (lines 28, 30, 32, 34, 36) is not an instruction, but denotes a 16 bit literal as an operand for the previous instruction.

Line	Address	Instr.	Oper.	
27	00200	LRHC	SIZE	; INIT. SIZE CONSTANT
				REGISTER
28	00201	LIT	SZE	; SIZE END ADDRESS
29	00202	LRHC	ITER	; SET TOTAL NUMBER OF
				ITERATIONS +1
30	00203	LIT	D#11	; 10 ITERATIONS
31	00204	LRHC	R5	; LOAD R5 WITH LAST, WHICH
				IS A CEILING
32	00205	LIT	LAST	; ON THE HIGHEST PRIME
				FACTOR
33	00206	LRHC	FLAG	; LOAD FLAG REGISTER
34	00207	LIT	FLG	; WITH FLAG CHARACTER
35	00208	LRHC	I	; LOAD J1 WITH LOOP3
				FOR LONG
36	00209	LIT	LOOP3	; CONDITIONAL JUMP
37	0020A	TOI	J1,I	

Figure 2. Initialize Registers block.

NOTE: Lines 1-26 contain the program header and the register definitions.

Following initialization, the iteration counter is decremented (Block 2). This is done by line 38 (address 020B hex). If the iteration counter does not equal zero, the program flow progresses to Block 3.

The primary function of Block 3 (Figure 3) is to clear the memory block which will be used as the flag array. It also clears registers which must be reset for each iteration of the Sieve routine, such as COUNT, the prime number count, and ADDR, the memory pointer for the flag array. The memory array is cleared using a delayed jump (DJIBOM) for maximum efficiency.

Line	Address	Instr.	Oper.	
40	0020D	LRHC	ADDR	; INIT. MEMORY POINTER
41	0020E	LIT	BASE	; BASE ADDRESS OF ARRAY
42	0020F	BEW	COUNT,COUNT	; CLEAR COUNT
43	00210	BEW	I,I	; CLEAR I
44	00211	CWU	ADDR,SIZE	; SET IA FLAGS FOR LOOP ENTRY
45	00212	LOOP2: DJIBOM	LT,(\$-LOOP2)	; LOOP UNTIL ALL FLAGS ARE CLEARED
46	00213	SA	H#FADDR	; CLEAR MEMORY LOCATION
47	00214	CWU	ADDR,SIZE	; SET IA FLAGS
48	00215	LRHC	ADDR	; RESET ADDR POINTER
49	00216	LIT	BASE	; TO BASE VALUE.

Figure 3. Clear Flag Array block.

Block 4 (Figure 4) fetches a new array word from memory and increments ADDR using the LFA (Load, Fetch and Augment) command. The code then checks to see if the end of the array is reached. If not, the code determines whether the memory address contains a flag (denoting a non-prime number) or zero (denoting a prime number). If a non-prime is found, Block 4 is repeated.

Line	Address	Instr.	Oper.	
50	00217	LOOP3: LFA	ADDR,ADDR	; RETRIEVE LOCATION ADDR
51	00218	LOOP4: CWU	ADDR,SIZE	; LOOK FOR END OF ARRAY
52	00219	RCV	WORD	; RETRIEVE WORD FROM P-M BUS
53	0021A	JIBOM	GT,(\$-LOOP1)	; EXIT IF ADDR>SIZE
54	0021B	CWU	WORD,FLAG	; SEE IF WORD=FLAG
55	0021C	DJIBOM	EQ,(\$-LOOP4)	; LOOP IF EQUAL
56	0021D	AWL	I,H#1	; INCREMENT I
57	0021E	LFA	ADDR,ADDR	; FETCH NEXT WORD

Figure 4. Load Word and Increment Pointer block.

If a prime number is found, Block 5 (Figure 5) is executed. In addition to incrementing the prime number count (COUNT), Block 5 calculates the actual value of the prime represented by the memory location containing the zero. Following Block 5, the memory pointer

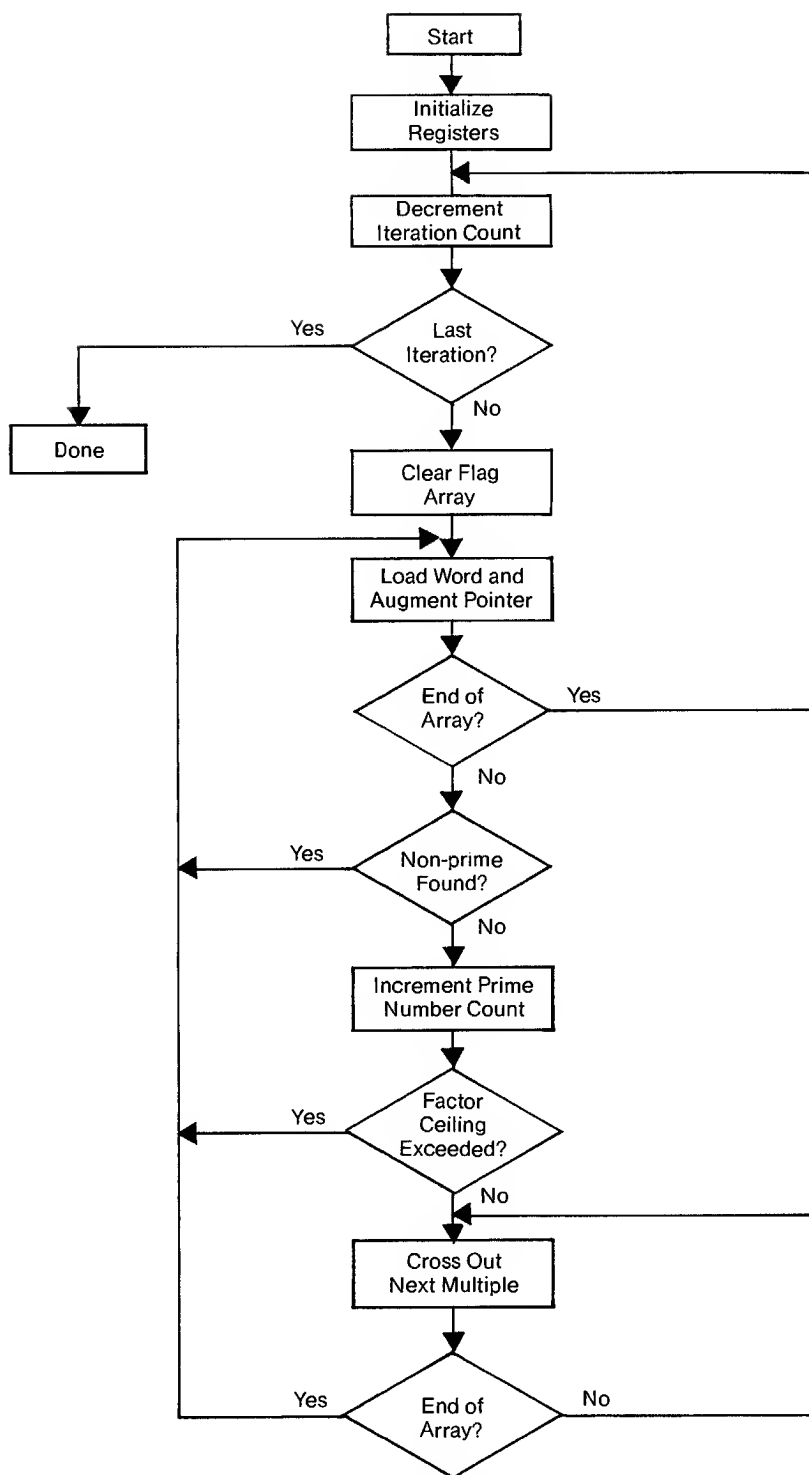


Figure 1. Word-flag program flow.

is compared with LAST. LAST represents the largest prime number whose multiples will result in the elimination of non-primes within the array. If ADDR is greater than LAST, Block 6 is bypassed. The comparison of LAST with ADDR is optional, and results in a performance improvement of 28 percent.

Line	Address	Instr.	Oper.	
58	0021F	CWU	I,R5	; LOOK FOR PRIME FACTOR CEILING
59	00220	DJIBOM	GT,(\$-LOOP3)	; IF CEILING, DON'T BOTHER
60				; TRYING TO CROSS ANYTHING OUT.
61				; LOOK FOR NEXT PRIME.
62	00221	AWL	COUNT,H#1	; INCREMENT PRIME NUMBER COUNT.
63	00222	SWL	ADDR,H#4	; COMPENSATE FOR EXTRA AUGMENT.
64	00223	LFD	EXOUT,ADDR	; LOAD DECREMENTED ADDR INTO EXOUT
65	00224	SWLL	I,PRIME	; PRIME = 2I + 1
66	00225	AWL	PRIME,H#1	;
67	00226	SWLL	PRIME,PRIME	; MULTIPLY PRIME BY FOUR
68	00227	SWLL	PRIME,PRIME	;

Figure 5. Increment Prime Number Count block.

When a prime number is found whose multiples may result in the flagging of non-primes, Block 6 (Figure 6) is executed. Block 6 uses a pointer denoted as EXOUT. EXOUT is initialized to the current value of (ADDR-4) before Block 6 is entered. It is incremented by PRIME, the actual value of the detected prime number, and a flag is stored at this address. This sequence is continued until EXOUT exceeds the ceiling of the array. Thus, all multiples of the detected prime number within the array are flagged. Following the execution of Block 6, the program proceeds back to Block 4.

Line	Address	Instr.	Oper.	
69	00228	LOOP5: CWU	EXOUT,SIZE	; CHECK FOR END OF ARRAY
70	00229	RIBO	J1,GT	; EXIT IF END
71	0022A	DJRM	(\$-LOOP5)	; LOOP UNTIL ALL FLAGS ARE SET
72	0022B	S	H#F,EXOUT	; SAVE FLAG IN MEMORY
73	0022C	AW	EXOUT,PRIME	; AUGMENT EXOUT BY 4*PRIME

Figure 6. Cross Out Next Multiple block.

## Execution Characteristics

Figure 7 illustrates the frequency of execution of each of the functional blocks. This analysis is very useful when optimizing the performance of the microcode routine. By moving instructions from frequently executed routines to those which are scarcely utilized, significant improvements in performance can be obtained.

Block 1 initializes the contents of certain registers and is executed only once. Block 2 and Block 3 are executed once for each iteration of the Sieve algorithm. These blocks decrement the iteration count, clear the flag array, and initialize registers which must be reset for each Sieve iteration.

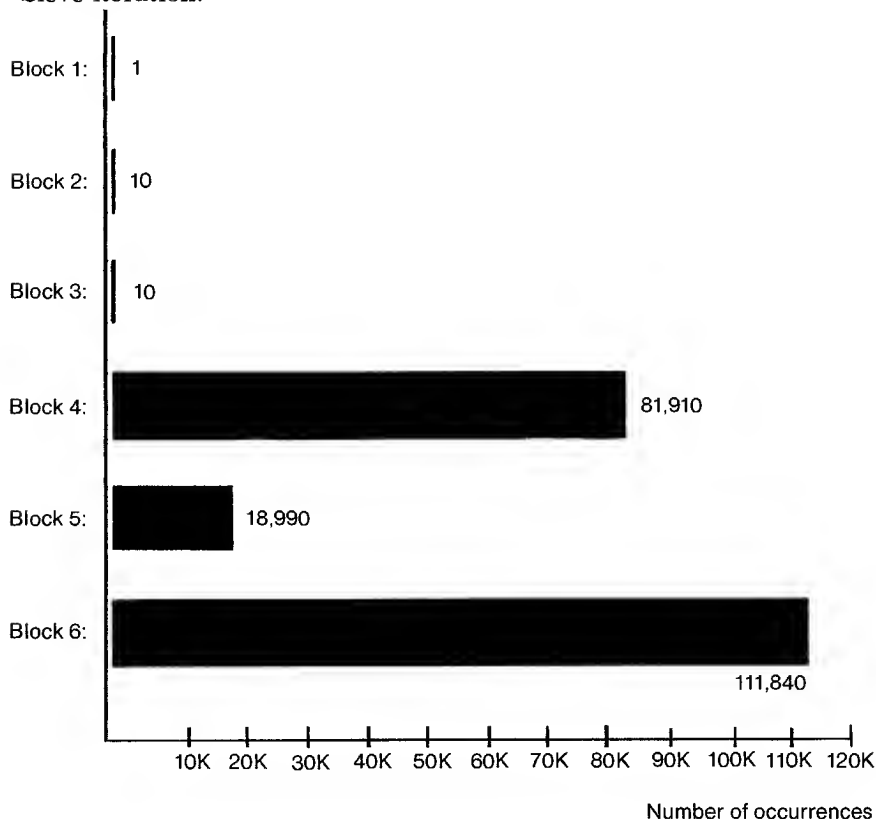


Figure 7. Block execution frequency.



Block 4 executes 81,910 times. This value corresponds directly to the dimension of the flag array in memory. The array contains 8190 flags, and the routine fetches an extra word to fulfill the loop's exit condition. The Sieve is executed ten times, yielding the expected value.

Block 5 executes 18,990 times. This value is equal to the actual number of primes within the array, multiplied by ten iterations.

Block 6 is the most frequently executed block. Since the number of multiples crossed out in one iteration (11,184) is greater than the number of non-primes within the array, many non-primes are flagged more than once.

From Figure 7 it is obvious that Blocks 4 and 6 must perform their function as efficiently as possible. This fact is further illustrated by Figure 8, which specifies the actual percentage of the total execution time occupied by each block. Note that Block 3 occupies a significant percentage of the total execution time due to an internal loop which iterates 8191 times for each iteration of the Sieve.

## **Program Optimization**

Since the NCR/32 is programmable at the microcode level, many aspects of program optimization do not become obvious through code inspection. Microcoding offers greater control of program execution to the programmer due to the lack of automatic sequencing by an internal microinstruction store.

The most effective optimization tools within the NCR/32 instruction set are the delayed jump instructions. Since the NCR/32 has a three stage internal instruction pipeline, the delayed jumps allow the instructions in the interpret and execute stages of the pipeline to execute before the jump is taken. Regular jump instructions simply nullify these two instructions, resulting in two machine cycles where no code is executed.

Delayed jumps are used in lines 45, 55, 59, and 71 of this routine. The use of these instructions saves two cycles each time the jump is taken, and thus results in performance improvements of 163,800; 125,840; 36,680; and 223,680 machine cycles. The total improvement of 550,000 machine cycles represents approximately one third of the measured total of 1,543,000 machine cycles for the Sieve benchmark.

Performance gains can also be realized by inserting a single-cycle instruction (which does not use the PM Bus) between the fetch (LFA) and receive (RCV) commands. This is allowed because the valid data is not presented on the PM Bus by the present memory interface until two cycles after the fetch was initiated. This technique is used in line 51 and results in a performance improvement of 81,910 machine cycles.

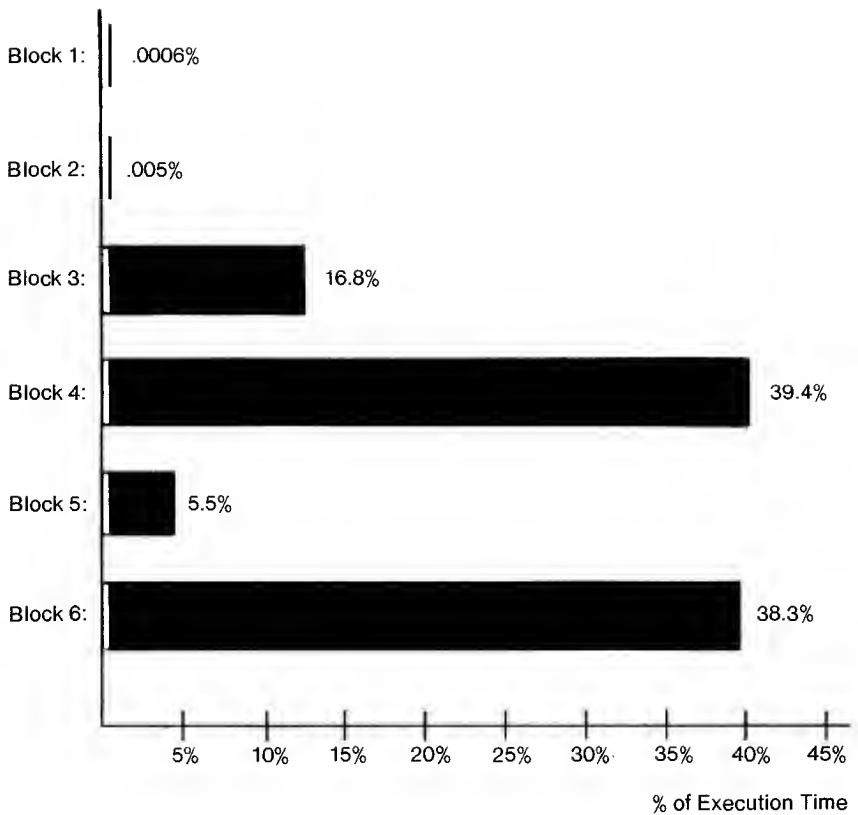


Figure 8. Percentage of execution time.

Program performance is also enhanced by removing instructions from frequently executed loops and placing them in other locations in the program. For instance, the jump from Block 6 to Block 4 requires a conditional jump of greater than 16 locations. This jump may be implemented using the JIBOL command, which requires a trailing literal. However, by loading a jump register with the desired jump address during Block 1 (which is executed once), the RIBO instruction can be used, shortening the loop (lines 69-73) by one instruction. The resulting performance improvement is 111,840 machine cycles.

## **“BYTE-FLAG” APPROACH**

The second approach to the Sieve algorithm is characterized by the use of 8-bit rather than 32-bit flags. This approach is aided by the field commands of the NCR/32 instruction set. The higher performance of the byte-flag approach is a result of the reduction in the number of memory accesses.

### **Field Commands**

The NCR/32 field commands facilitate the processing of large (1 to 64k-1) fields composed of bytes. The instructions operate on one, two, or three fields simultaneously, and remain frozen in the instruction pipeline until either a word boundary is crossed or the Tally register equals zero. The specific field instruction used in the byte-flag implementation of the Sieve is the CBFU (Compare Byte to Field Unsigned) command, which is described later in this text.

### **Program Flow**

Program flow for the byte-flag approach is illustrated in Figure 9. A comparison with the word-flag flow (Figure 1) reveals that the byte-flag approach uses a larger number of functional blocks and a more complex control scheme. This is due in part to the complexity of the field commands.

Again, the functional blocks are assigned the following numbers for ease of referral:

- Block 1: Initialize Registers.
- Block 2: Decrement Iteration Count.
- Block 3: Clear Flag Array.
- Block 4: Load Word and Augment Pointer.
- Block 5: Field Compare.
- Block 6: Overflow Routine.
- Block 7: Increment Prime Number Count.
- Block 8: Cross Out Next Multiple.

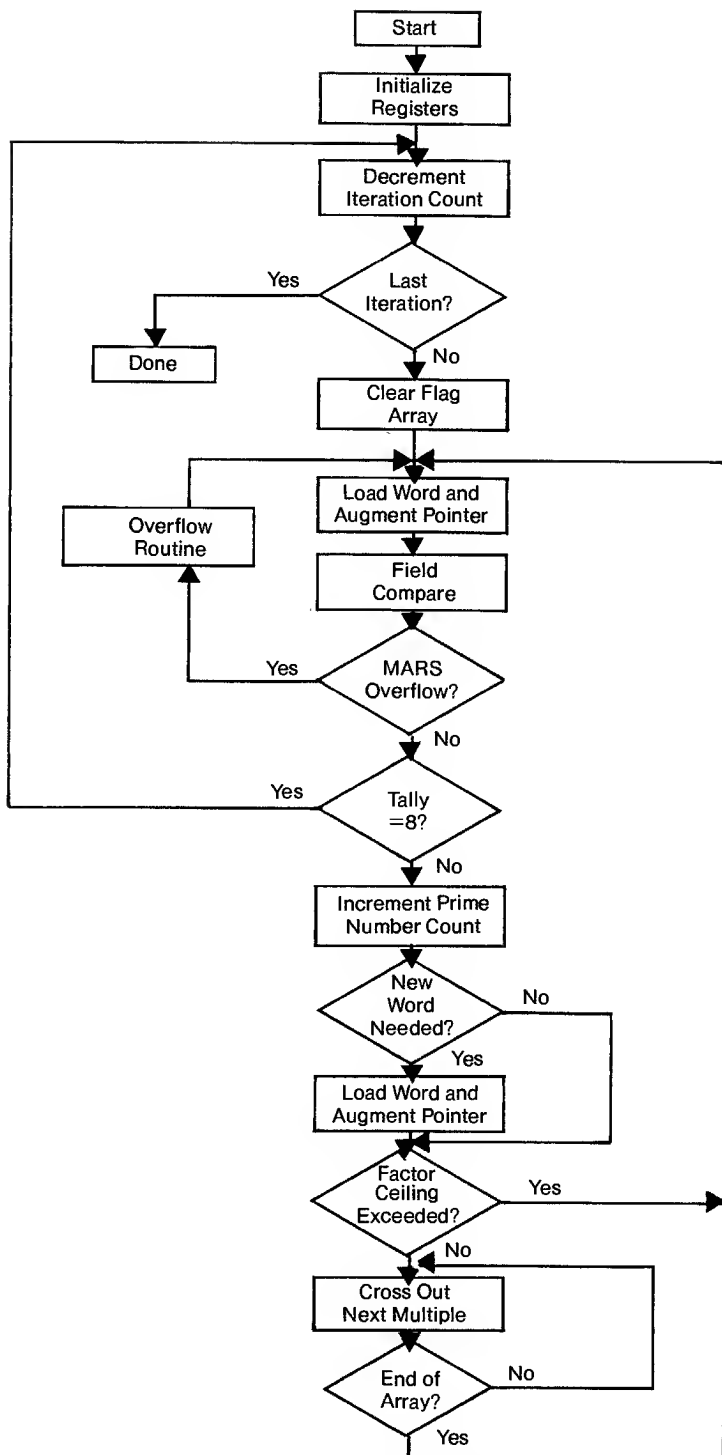


Figure 9. Byte-flag program flow.

Blocks 1 through 4 perform virtually the same function in the byte-flag approach as in the word-flag approach. However, additional initialization is needed to load the Tally register and various jump registers. Additionally, the flag value must be stored in two registers to accommodate the field commands.

Note that in lines 74-75, ADDR is initialized to BASE+1 rather than to BASE. If this is not done, then nine will be counted as a prime number. This occurs because the flag representing the number nine will already be contained in the WORD register when that location in memory is flagged as non-prime. Initializing ADDR to BASE+1 causes the first WORD of the array to contain flags representing "x 3 5 7" rather than "3 5 7 9."

Once the memory word is loaded, Block 5, the field compare, is executed. Block 5 consists of the CBFU command. The flow for this command is illustrated in Figure 10.

The CBFU command compares a word to a target byte, one byte at a time. The field byte is selected from the word in the MARS5 data register by the MARS5 byte pointers. The CBFU command holds in the execution stage of the instruction pipeline until either the Tally register equals zero, a field byte does not match the target byte, or a MARS5 overflow is detected. An overflow occurs when the MARS5 data register byte pointers (the two low order bits of R11) are incremented beyond the boundary of word specified by the MARS5 address register (R10). Thus, if the two low order bits of R11 are both equal to one and the byte pointers are incremented, an overflow occurs.

Following the field compare, the JFA command is executed. This jump does not occur unless a MARS overflow flag is set. If a flag is set, the Field Array Bits (bits 1-5) of the State Register (IRU9) are left justified, zero filled, and concatenated with the most significant byte of Jump Register 7 to form the jump address. Thus, a MARS5 overflow will jump to a specific routine designed to handle that overflow.

Block 6 (Figure 11) is the MARS5 Overflow Routine. This routine clears the overflow flag in the State Register, loads a new word from memory, and augments the memory pointer. At the conclusion of the overflow routine, the program loops back to the field compare (Block 5).

Line	Address	Instr.	Oper.	
105	00310	M50VF: ORG	H#310	; OVERFLOW ROUTINE
106	00310	BEW	I,I	;
107	00311	TOI	STREG,I	; CLEAR OVERFLOW FLAGS
108	00312	DRIBZ	J1,UN	; DLY'D RETURN TO LOOP4
109	00313	LFA	ADDR,ADDR	; FETCH NEW WORD AND RETURN
110	00314	RCV	WORD	

Figure 11. MARS5 Overflow Routine.

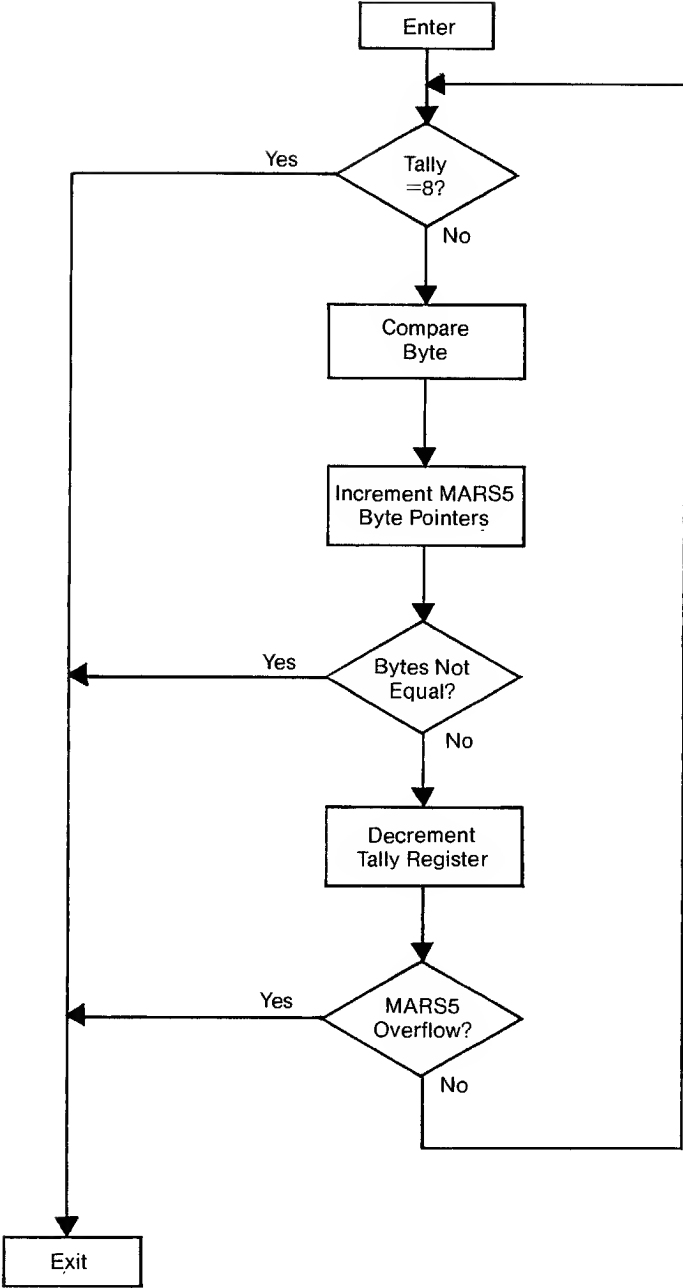


Figure 10. CBFU command flow.

If the field compare is halted and an overflow is not detected, a byte has been found which does not match the target byte. This indicates that the number represented by that byte is a prime number. In this case, Block 7 is executed.

Block 7 (Figure 12) performs virtually the same function as Block 5 of the word-flag approach. However, the value of the prime number is calculated based on the value of the Tally register rather than on a counting register.

Line	Address	Instr.	Oper.	
80	00226	TII	TALLY,I	; RETRIEVE TALLY VALUE
81	00227	BAW	I,I	; SEE IF TALLY=0
82	00228	RIBO	J2,Z	; IF SO, NEXT ITERATION
83	00229	SRB3O	H#0,ADDR	; LOAD ONLY IF NECESSARY
84	0022A	LFA	ADDR,ADDR	
85	0022B	SRB3O	H#0,ADDR	; RCV ONLY IF NECESSARY
86	0022C	RCV	WORD	
87	0022D	LSTCHK: CWU	I,LAST	; LOOK FOR LAST FACTOR WHICH
88				; WILL EX OUT NON-PRIMES.
89	0022E	DJIBOM	LT,(\$-LOOP4)	; IF FOUND, FIELD COMPARE
90	0022F	AWL	COUNT,H#1	; INCREMENT PRIME COUNT
91	00230	LIT	H#AAAA	; NO-OP

Figure 12. Increment Prime Number Count block.

Block 8 (Figure 13) performs the same function as Block 6 of the word-flag approach. However, since the flags are being stored at byte rather than word addresses, the TBF (Transfer Byte to Field) command is necessary to set the MARS6 Byte Write Tags. Also, since MARS6 is the only MARS pair which automatically supplies Memory Write Tags, MARS6 Data (R13) must be loaded (during initialization) with 4 flag bytes.

Line	Address	Instr.	Oper.	
98	00237	LOOP5: AW	EXOUT,PRIME	; AUGMENT EXOUT BY PRIME
99	00238	CWU	EXOUT,SIZE	; CHECK FOR END OF ARRAY
100	00239	DJIBOM	H#0,(\$-LOOP5)	; LOOP IF NOT END
101	0023A	TBF	R3B3	; SET MARS6 WRITE TAG REGISTER
102	0023B	S	H#0,EXOUT	; SAVE FLAG IN MEMORY

Figure 13. Cross Out Next Multiple block.

## Execution Characteristics

Figure 14 is a histogram depicting the execution frequency of the blocks defined above. It is interesting to compare this figure with the comparable histogram (Figure 7) for the word-flag approach. Note that the byte-flag approach reduces the number of Block 4 (load word and augment pointer) iterations to one quarter of the original value. The other significant difference between the two approaches is the addition of the field compare (Block 5; 96,240 iterations) in the byte-flag program.

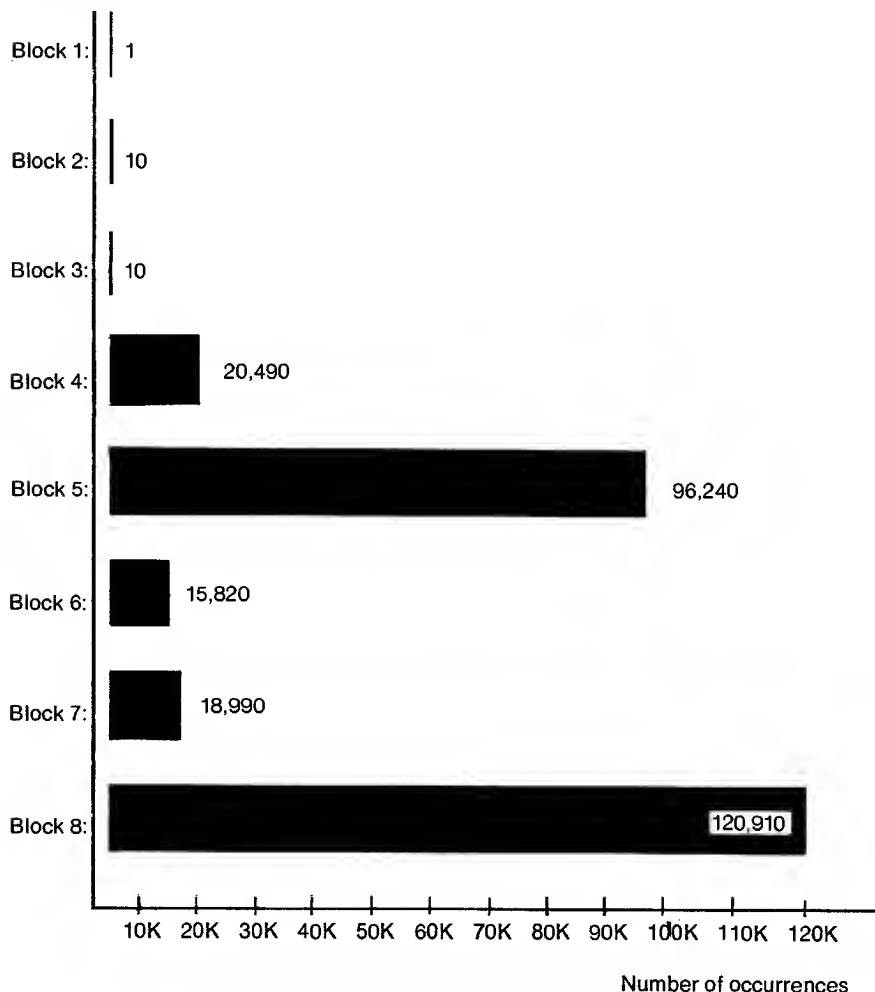


Figure 14. Block execution frequency.



A comparison of the execution time percentages (Figures 8 and 15) reveals the greater efficiency of the byte-flag approach. In this approach, flagging of the non-prime numbers in the array accounts for more than fifty percent of total execution time. This percentage is so high for two reasons. Block 8 in the byte-flag approach is one step longer than Block 6 in the word-flag approach, due to the necessary inclusion of the TBFU command which sets the MARS6 Write Tags. The percentage is also higher due to the reduction in the number of fetches (a three cycle operation) by a factor of four.

Block 5, the block which had the second highest number of iterations, only consumes 8.4 percent of the total execution time, since each iteration takes only one clock cycle. The Increment Prime Number Count block increases from 5.5 percent of total execution time in the word-flag approach to twenty percent due to the more complex control needed for the byte-flag approach. This control function includes the determination of whether a new memory word must be fetched when a word boundary is reached but no overflow occurs.

## **Program Optimization**

Program optimization is largely achieved using the same techniques described above for the word-flag approach. All of the blocks except for Block 7 were, in fact, easily reduced to their present, optimized form. The primary hurdle presented by Block 7 was the need to retrieve a new memory word if the ADDR pointer had reached a word boundary and a prime number simultaneously.

This problem was handled by using the SRB30 command. This command skips a step based on the value of a bit pair in the least significant byte of any register. Since a word boundary is indicated when the two low order bits of ADDR are both zero, the LFA and RCV commands are skipped if either of these bits is set. See Figure 12, lines 83-86.

## **NCR/32 PERFORMANCE RESULTS**

Figure 16 illustrates the execution performance of four versions of the Sieve algorithm. Versions 0 and 1 utilize the word-flag approach, and Version 1 also uses the algorithm improvement mentioned earlier in this text. Similarly, Versions 2 and 3 use the byte-flag approach, and Version 3 uses the algorithm improvement.

The effectiveness of the algorithm improvement is shown in Figure 17. For the word-flag approach the improvement is 28 percent, while for the byte-flag approach it is 35 percent.

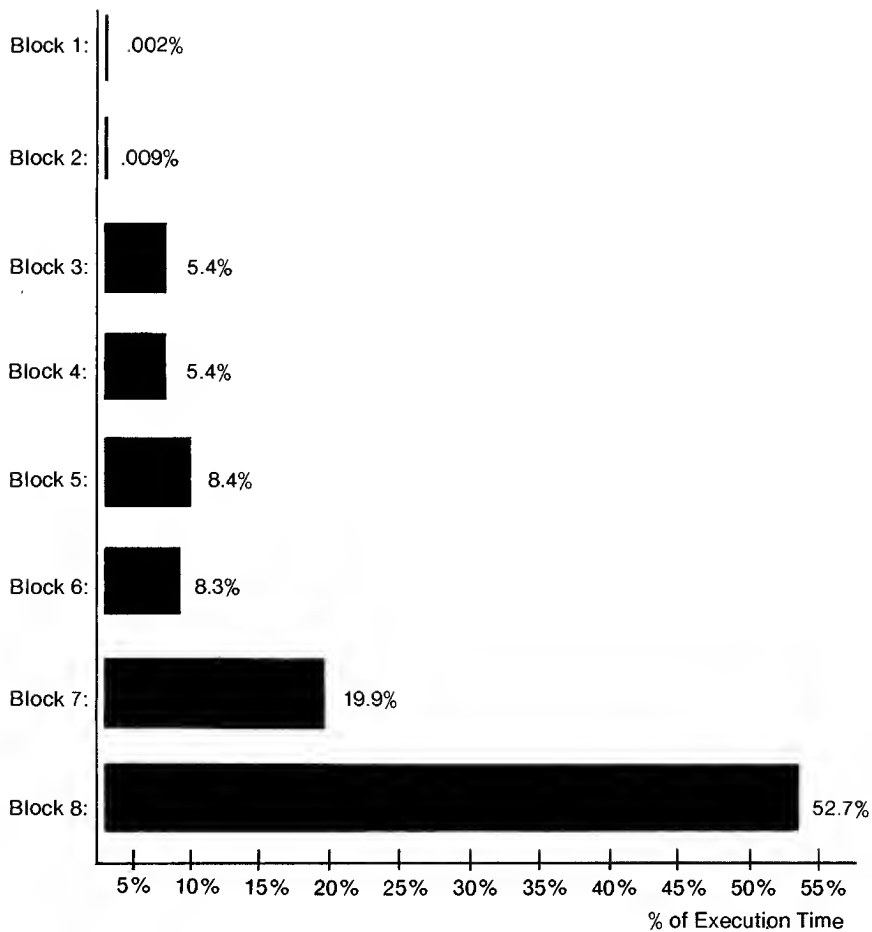


Figure 15. Percentage of execution time.

Version	Cycles	Characteristics
0	1,976,000	Word-flag, factor ceiling not used
1	1,545,000	Word-flag, factor ceiling used
2	1,600,000	Byte-flag, factor ceiling not used
3	1,187,000	Byte-flag, factor ceiling used

Figure 16. Number of execution cycles.

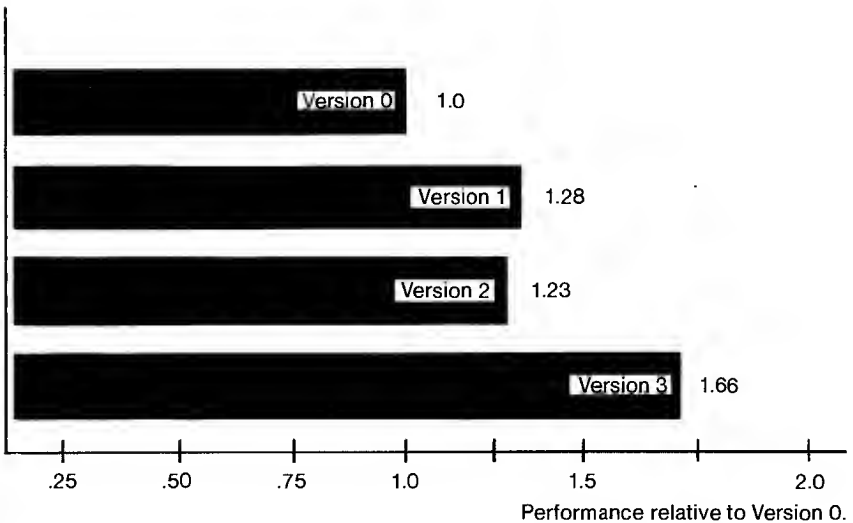


Figure 17. Program Performance.

Processor	Clock Speed	Execution Time
6809	*	5.1 seconds
68000	8 MHz	0.49 seconds
8086	8 MHz	1.90 seconds
8088	5 MHz	4.0 seconds
NCR/32	13.3 MHz	0.18 seconds
TM-16 (bit slice)	*	1.98 seconds
Z80	*	6.8 seconds
Z8001	*	1.73 seconds

\*denotes information not published in the January 1983 BYTE Magazine article.

Figure 18. Comparative performance table.

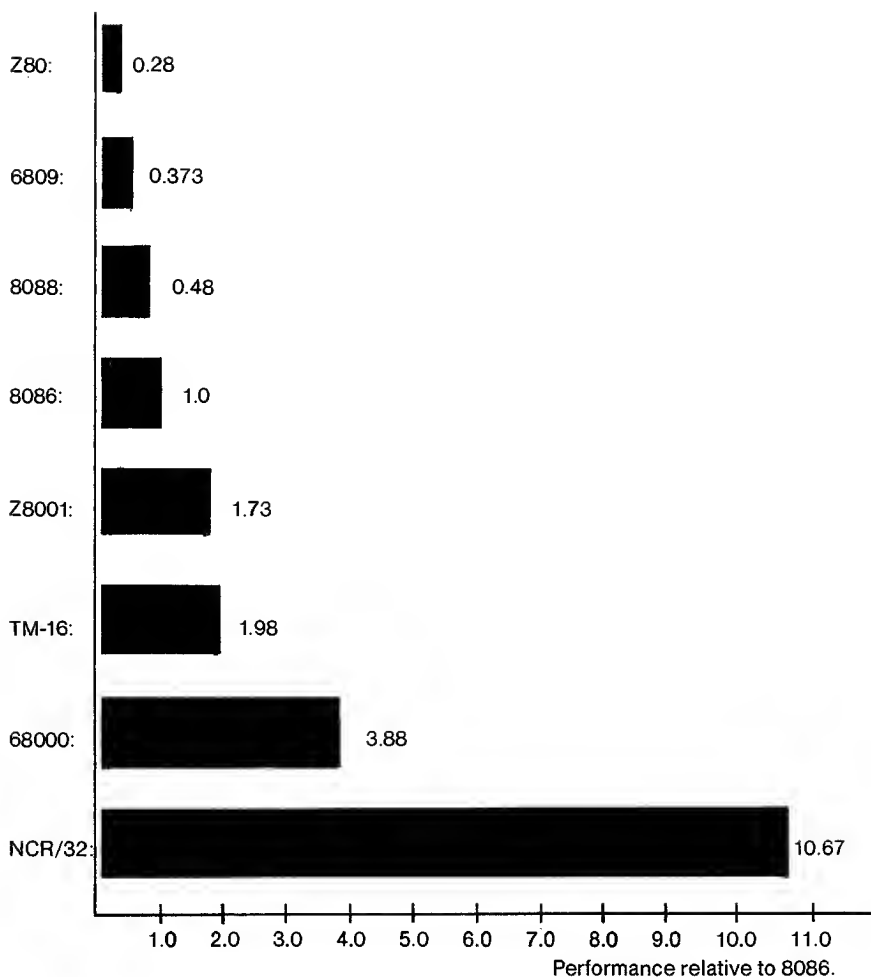


Figure 19. Processor Performance Comparison.

With a 150 nanosecond clock cycle, the fastest version of the Sieve runs on the NCR/32 in 0.178 seconds. Figure 18 compares this result with those published in BYTE magazine for other commercial microprocessors. This comparison is illustrated graphically in Figure 19. As shown in the histogram, the NCR/32 is 2.75 times faster than the second fastest processor.

## REFERENCES

The following documents were used in the development of this application note:

“Eratosthenes Revisited: Once More through the Sieve.” Jim Gilbreath, BYTE Magazine, January, 1983.

*NCR/32 General Information Manual*. NCR Corporation, Colorado Springs, 1983.

NCR/32-000 32-Bit Microprogrammable Microprocessor Data Sheet. NCR Corporation, Colorado Springs, 1983.

## CODE LISTINGS

Below are the code listings for versions 1 and 3 of the Sieve program. Version 1 uses the word-flag approach, and Version 3 uses the byte-flag approach. Both versions include the algorithm improvement.

For complete definitions of the instructions and the chip itself, refer to the NCR/32 General Information Manual and the NCR/32-000 Central Processor Chip data sheet.

1	;		PROGRAM:	SIEVE OF ERATOSTHENES BENCHMARK
2	;			
3	;		PROGRAMMER:	JOHN BEEKLEY
4	;			
5	;		DATE:	OCTOBER 18, 1983
6	;			
7	;		UPDATED:	JANUARY 3, 1984
8	;			
9	;			
10	;			
11		START: EQU H#200		; STARTING ADDRESS OF SUBROUTINE
12		BASE: EQU START+H#100		; BASE ADDRESS FOR ARRAY
13		SZE: EQU BASE+D#32760		; END ADDRESS OF ARRAY (8190*4)
14		FLG: EQU H#FFFF		; FLAG FOR FACTORS
15		LAST: EQU D#65		; CEILING ON FACTORS: SQR(16361)/2
16		WORD: EQU R11		; CURRENT BYTES BEING CHECKED
17		FLAG: EQU R9		; FLAG CONSTANT
18		I: EQU R13		; CURRENT BYTE NUMBER
19		PRIME: EQU R2		; VALUE OF CURRENT PRIME NUMBER
20		COUNT: EQU R3		; NUMBER OF PRIMES FOUND
21		ITER: EQU R4		; NUMBER OF ITERATIONS
22		EXOUT: EQU R8		; NONPRIME FACTOR ELIMINATION
23		SIZE: EQU R7		; SIZE CONSTANT FOR ENDPOINT CHECK
24		ADDR: EQU R12		; MEMORY ADDRESS POINTER
25				
26	00200	SIEVE: ORG START		; START OF ROUTINE
27	00200	LRHC SIZE		; INIT. SIZE CONSTANT REGISTER
28	00201	LIT SZE		; SIZE END ADDRESS
29	00202	LRHC ITER		; SET TOTAL NUMBER OF ITERATIONS +1
30	00203	LIT D#11		; 10 ITERATIONS
31	00204	LRHC R5		; LOAD R5 WITH LAST, WHICH IS A CEILING
32	00205	LIT LAST		; ON THE HIGHEST PRIME FACTOR
33	00206	LRHC FLAG		; LOAD FLAG REGISTER
34	00207	LIT FLG		; WITH FLAG CHARACTER

35	00208	LRHC	I	; LOAD J1 WITH LOOP3
36	00209	LIT	LOOP3	FOR LONG
37	0020A	TOI	J1,I	; CONDITIONAL JUMP
38	0020B	LOOP1: SWL	ITER,H#1	; DECREMENT ITERATION
39	0020C	RIBO	J0,Z	COUNT
40	0020D	LRHC	ADDR	; RETURN IF TEN ITERATIONS
41	0020E	LIT	BASE	; INIT. MEMORY POINTER
42	0020F	BEW	COUNT,COUNT	; BASE ADDRESS OF ARRAY
43	00210	BEW	I,I	; CLEAR COUNT
44	00211	CWU	ADDR,SIZE	; CLEAR I
				; SET IA FLAGS FOR LOOP
				ENTRY
45	00212	LOOP2: DJIBOM	LT,(\$-LOOP2)	; LOOP UNTIL ALL FLAGS
				ARE CLEARED
46	00213	SA	H#F,ADDR	; CLEAR MEMORY LOCATION
47	00214	CWU	ADDR,SIZE	; SET IA FLAGS
48	00215	LRHC	ADDR	; RESET ADDR POINTER
49	00216	LIT	BASE	; TO BASE VALUE.
50	00217	LOOP3: LFA	ADDR,ADDR	; RETRIEVE LOCATION ADDR
51	00218	LOOP4: CWU	ADDR,SIZE	; LOOK FOR END OF ARRAY
52	00219	RCV	WORD	; RETRIEVE WORD FROM P-M
				BUS
53	0021A	JIBOM	GT,(\$-LOOP1)	; EXIT IF ADDR>SIZE
54	0021B	CWU	WORD,FLAG	; SEE IF WORD=FLAG
55	0021C	DJIBOM	EQ,(\$-LOOP4)	; LOOP IF EQUAL
56	0021D	AWL	I,H#1	; INCREMENT I
57	0021E	LFA	ADDR,ADDR	; FETCH NEXT WORD
58	0021F	CWU	I,R5	; LOOK FOR PRIME FACTOR
				CEILING
59	00220	DJIBOM	GT,(\$-LOOP3)	; IF CEILING, DON'T BOTHER
60				; TRYING TO CROSS
				ANYTHING OUT
61				; LOOK FOR NEXT PRIME.
62	00221	AWL	COUNT,H#1	; INCREMENT PRIME
				NUMBER COUNT.
63	00222	SWL	ADDR,H#4	; COMPENSATE FOR EXTRA
				AUGMENT.
64	00223	LFD	EXOUT,ADDR	; LOAD DECREMENTED ADDR
				INTO EXOUT
65	00224	SWLL	I,PRIME	; PRIME = 2I + 1
66	00225	AWL	PRIME,H#1	;
67	00226	SWLL	PRIME,PRIME	; MULTIPLY PRIME BY FOUR
68	00227	SWLL	PRIME,PRIME	;
69	00228	LOOP5: CWU	EXOUT,SIZE	; CHECK FOR END OF ARRAY
70	00229	RIBO	J1,GT	; EXIT IF END
71	0022A	DJRM	(\$-LOOP5)	; LOOP UNTIL ALL FLAGS
				ARE SET
72	0022B	S	H#F,EXOUT	; SAVE FLAG IN MEMORY
73	0022C	AW	EXOUT,PRIME	; AUGMENT EXOUT BY
				4*PRIME
74	END			

```

1      TITLE          SIEVE OF ERATOSTHENES BENCHMARK, V.3
2      ;
3      ;
4      ;              PROGRAM          SIEVE OF ERATOSTHENES
5      ;                      BENCHMARK
6      ;                      VERSION 3: USING FIELD
7      ;                      COMMANDS
8      ;                      AND FACTOR CEILING
9      ;
10     ;              PROGRAMMER:      JOHN BEEKLEY
11     ;
12     ;              DATE:            DECEMBER 1, 1983
13     ;
14     ;              UPDATED:         JUNE 1, 1984
15     ;
16     START:  EQU      H#200          ; STARTING ADDRESS OF
17     DIM:    EQU      D#8191         ; SUBROUTINE
18     BASE:   EQU      H#0000         ; DIMENSION OF MEMORY
19     SIZE:   EQU      BASE+DIM+1     ; ARRAY
20     FLG:    EQU      H#FFFF         ; BASE ADDRESS FOR ARRAY
21     LST:    EQU      DIM-D#65       ; END ADDRESS OF ARRAY
22     ;                                ; FLAG FOR FACTORS
23     ;                                ; LAST FACTOR WHICH
24     ;                                ; WILL CAUSE
25     ;                                ; NON-PRIMES TO BE
26     ;                                ; CROSSED
27     ;                                ; OUT. D#65 IS AN
28     ;                                ; APPROXIMATION
29     ;                                ; OF SQR(16361)/2
30     LAST:   EQU      R5             ; STORE LST IN R5
31     WORD:   EQU      R11            ; CURRENT BYTES BEING
32     ;                                ; CHECKED
33     FLAG:   EQU      R9             ; FLAG CONSTANT
34     FLAG2:  EQU      R13            ; FLAG FOR STORAGE IN
35     ;                                ; MEMORY
36     I:      EQU      R1             ; CURRENT BYTE NUMBER
37     I:      EQU      R1B3           ; I'S LEAST SIGNIFICANT BYTE
38     PRIME:  EQU      R2             ; VALUE OF CURRENT
39     ;                                ; PRIME NUMBER
40     TSTBYT: EQU      R2B3           ; TEST BYTE
41     COUNT:  EQU      R15            ; NUMBER OF PRIMES FOUND
42     ITER:   EQU      R4             ; NUMBER OF ITERATIONS
43     EXOUT:  EQU      R12            ; NONPRIME FACTOR
44     ;                                ; ELIMINATION
45     SIZE:   EQU      R7             ; SIZE CONSTANT FOR
46     ;                                ; ENDPOINT CHECK
47     ADDR:   EQU      R10            ; MEMORY ADDRESS POINTER
48     TSTORE: EQU      R0             ; TALLY REGISTER STORAGE
49     FLDFLG: EQU      R3             ; FLAG BYTE FOR WRITE TAG
50     ;                                ; GENERATION
51     00200  SIEVE:  ORG      START    ; START OF ROUTINE
52     00200  LRHC   ITER             ; SET TOTAL NUMBER OF
53     00201  LIT    D#11             ; ITERATIONS +1
54     ;                                ; 10 ITERATIONS

```



44	00202	LRHC	SIZE	; INIT. SIZE CONSTANT REGISTER
45	00203	LIT	SIZE	; SIZE END ADDRESS
46	00204	LRHC	FLAG	; LOAD FLAG REGISTER
47	00205	LIT	FLG	; WITH FLAG CHARACTER
48	00206	TRHLH	FLAG,FLAG	
49	00207	TW	FLAG,FLDFLG	; LOAD AUXILLIARY FLAG REGISTER
50	00208	LRHC	LAST	; LOAD LAST FACTOR REGISTER
51	00209	LIT	LST	; WITH LST
52	0020A	LRHC	I	
53	0020B	LIT	H#300	; LOAD J7 FOR FIELD ARRAY JUMPS
54	0020C	TOI	J7,I	
55	0020D	LRHC	I	; LOAD J1 WITH LOOP4
56	0020E	LIT	LOOP4	
57	0020F	TOI	J1,I	
58	00210	LRHC	I	; LOAD J2 WITH LOOP1
59	00211	LIT	LOOP1	
60	00212	TOI	J2,I	
61	00213	LOOP1: SWL	ITER,H#1	; DECREMENT ITERATION COUNT
62	00214	RIBO	J0,2	; RETURN IF TEN ITERATIONS
63	00215	LRHC	TSTORE	; LOAD TALLY REGISTER WITH DIM
64	00216	LIT	DIM	
65	00217	TOI	TALLY,STORE	
66	00218	LRHC	ADDR	; INIT. MEMORY POINTER
67	00219	LIT	BASE+1	; BASE ADDRESS OF ARRAY
68	0021A	BEW	COUNT,COUNT	; CLEAR COUNT
69	0021B	BEW	WORD,WORD	; CLEAR WORD
70	0021C	CWU	ADDR,SIZE	; SET IA FLAGS FOR LOOP ENTRY
71	0021D	LOOP2: DJIBOM	LT,(\$-LOOP2)	; LOOP UNTIL ALL FLAGS ARE CLEARED
72	0021E	SA	H#F,ADDR	; CLEAR MEMORY LOCATION
73	0021F	CWU	ADDR,SIZE	; SET IA FLAGS
74	00220	LRHC	ADDR	; RESET ADDR POINTER
75	00221	LIT	BASE+1	; TO BASE VALUE.
76	00222	LOOP3: LFA	ADDR,ADDR	; RETRIEVE LOCATION ADDR
77	00223	RCV	WORD	; AND STORE IN WORD.
78	00224	LOOP4: CBFU		; COMPARE WORD TO FLAG BYTE
79	00225	JFA		; JUMP TO ROUTINE IF OVERFLOW
80	00226	TII	TALLY,I	; RETRIEVE TALLY VALUE
81	00227	BAW	I,I	; SEE IF TALLY=0
82	00228	RIBO	J2,Z	; IF SO, NEXT ITERATION
83	00229	SRB3O	H#0,ADDR	; LOAD ONLY IF NECESSARY
84	0022A	LFA	ADDR,ADDR	
85	0022B	SRB3O	H#0,ADDR	; RCV ONLY IF NECESSARY
86	0022C	RCV	WORD	
87	0022D	LSTCHK: CWU	I,LAST	; LOOK FOR LAST FACTOR WHICH
88				; WILL EX OUT NON-PRIMES.
89	0022E	DJIBOM	LT,(\$-LOOP4)	; IF FOUND, FIELD COMPARE
90	0022F	AWL	COUNT,H#1	; INCREMENT PRIME COUNT

91	00230	LIT	H#AAAA	; NO-OP
92	00231	SW	I,TSTORE	; CALCULATE BYTE NUMBER
93	00232	BIW	I,I	; CONTINUE CALCULATING BYTE #
94	00233	SWLL	I,PRIME	; CALCULATE PRIME NUMBER VALUE
95	00234	AWL	PRIME,H#3	; PRIME = 2I+1 = 2(I-1)+3
96	00235	TW	ADDR,EXOUT	; LOAD ADDR-5 INTO EXOUT AND BEGIN
97	00236	SWL	EXOUT,H#5	; XING OUT MULTIPLES OF PRIME.
98	00237	LOOP5: AW	EXOUT,PRIME	; AUGMENT EXOUT BY PRIME
99	00238	CWU	EXOUT,SIZE	; CHECK FOR END OF ARRAY
100	00239	DJIBOM	LT,(\$-LOOP5)	; LOOP IF NOT END
101	0023A	TBF	R3B3	; SET MARS6 WRITE TAG REGISTER
102	0023B	S	H#0,EXOUT	; SAVE FLAG IN MEMORY
103	0023C	JRM	(\$-LOOP4)	; RETURN TO FIELD COMPARE
104		; MARS5 OVERFLOW ROUTINE		
105	00310	M5OVF: ORG	H#310	; OVERFLOW ROUTINE
106	00310	BEW	I,I	;
107	00311	TOI	STREG,I	; CLEAR OVERFLOW FLAGS
108	00312	DRIBZ	J1,UN	; DLY'D RETURN TO LOOP4
109	00313	LFA	ADDR,ADDR	; FETCH NEW WORD AND RETURN
110	00314	RCV	WORD	
111	END			



VLSI Processor Products  
NCR Microelectronics Division  
Colorado Springs, Colorado  
(303) 596-5612  
(800) 525-2252  
Telex 452457

RM-0480  
ST-2104-23 0984